IMPLEMENTATION OF MULTI-ALGORITHM CONTROLLERS FOR PATH DETERMINATION IN MOBILE ROBOT SYSTEMS

by

Adrianus Victor Hitijahubessy, B.S.M.E.

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

The University of Texas at Austin

MAY 2006

IMPLEMENTATION OF MULTI-ALGORITHM CONTROLLERS FOR PATH DETERMINATION IN MOBILE ROBOT SYSTEMS

Approved by
Supervising Committee:

_____

_____

Dedication

To my parents for their continuous support, guidance, and prayers.

Abstract

IMPLEMENTATION OF MULTI-ALGORITHM CONTROLLERS FOR PATH DETERMINATION IN MOBILE ROBOT SYSTEMS

Adrianus Victor Hitijahubessy, M.S.E.

The University of Texas at Austin, 2006

Supervisor: Benito R. Fernandez

Recent advancements in control systems, such as the ones used in missile technology in the military or autonomous vehicle development have motivated this study in an attempt to explore various control algorithms and their implementation relevant those applications. Both missile interceptor and autonomous vehicle technology require precise and responsive control system to accurately determine the projectile path of pursuer to strike a moving target or reach a static finish line.

The objective of this study is to investigate the performance of several control techniques for a mobile robot to autonomously track and pursue a moving object. Computer model is developed to numerically predict the path taken by the pursuer as it tracks an object moving in regular or random manner. In the computer simulation, the

robot's path is calculated using three different techniques: reactive controller, linear estimation, and artificial neural network. Fitness of each method may be determined by evaluating the controller against several factors, such as interception time, steady-state positional error, steady-state time (settling time) and algorithm complexity, listed in decreasing order of importance.

A working experimental model is developed to validate the controller selection determined from the computer model simulation. In the experimental setting, the primary inputs to the robot are visual images from cameras. The experiments are carried out with the robot receiving visual inputs from two different perspectives, overhead and frontal vision. Robust image processing technique becomes a topic of significant importance for the system. To manipulate visual images in real-time from raw inputs to comprehensible data, while maintaining fast computational time is a challenge that is addressed in this study.

The results from computer simulations show that artificial neural network is a more powerful control algorithm, capable of estimating the object's path more accurately than the other two controllers, resulting in smaller steady-state positional error. The experimental results confirm this conclusion as artificial neural network outperforms the reactive and linear controller by intercepting the object more quickly, i.e. shorter interception time.

Table of Contents

List of Tables

List of Figures

Chapter 1 – Introduction

1.1. Application of Autonomous Control Systems

Autonomous Control System (ACS) is an independent control system that is capable of executing desired tasks in unstructured environments without human guidance in its operation [1]. An example of an application of ACS is in air defense where an anti-ballistic missile (ABM) is employed to intercept ballistic or cruise missiles. The ballistic missile follows a prescribed course which is generally fixed and not alterable, while cruise missile has the capability to maneuver after launch by means of jet propulsion or aerodynamics effect in general following the terrain at a prescribed altitude profile. When deployed in air combat, surface-to-air missiles (SAMs) need to have the flexibility to control and alter their pursuit path after released from ground launcher [2].

Another sample application is in the development of automatic guided vehicle (AGV) systems, where vehicle autonomously maneuvers itself in a three-dimensional environment to perform a desired task, for example an underwater ROVER [3]. The controller has to perform in real-time using inputs from various sensors attached to the vehicle. In 2004, the Defense Advanced Research Projects Agency (DARPA) launched for the first time an open competition DARPA Grand Challenge that aimed to develop fully autonomous vehicles capable of traveling across a 150-mile long desert [4]. The vehicles were allowed to have both on-board sensors and GPS capability.

1.2. RESEARCH GOALS AND METHODOLOGY

In the course of this study, the performance of several control techniques will be investigated and their fitness analyzed when applied in a specific environment to simulate the controlled dynamic of an anti-ballistic missile or autonomous vehicle system. The system under study is a mobile robot whose task is to autonomously track and pursue a moving object (target). The robot's control system is desired to have the ability to:

1. gather information from the environment via sensors,
2. navigate from point A to B, where point B may be a stationary finish point, or
3. track or intercept another robot in motion, and
4. operate autonomously, i.e. without human intervention.

For robustness, it is also desired that the robot have the ability to learn. Learning will enable to robot to adapt to surroundings and learn new strategies based on surrounding inputs without outside assistance.

The main sensor for the pursuing robot is the camera as a sensor, which is mounted overhead or on-board. The cameras are used to obtain positional information about both the pursuing robot (predator), with overhead camera, and pursued object (prey), with overhead and/or on-board cameras. The overhead camera is comparable to GPS tracking capability of vehicles in the DARPA Grand Challenge competition, or more appropriately called Local Positioning System (LPS) for a smaller-scale system. In navigating from point A to B, the pursuit path is calculated deterministically using three control techniques. In basic automatic controller, current positional error is calculated and used to determine the magnitude of adjustment needed to reduce error. The error will

be continuously monitored and fed back to controller which outputs correcting commands to actuators until desired state is achieved [5]. This is essentially a closed-loop control system. Block diagram representation is shown in Fig. 1 [6].

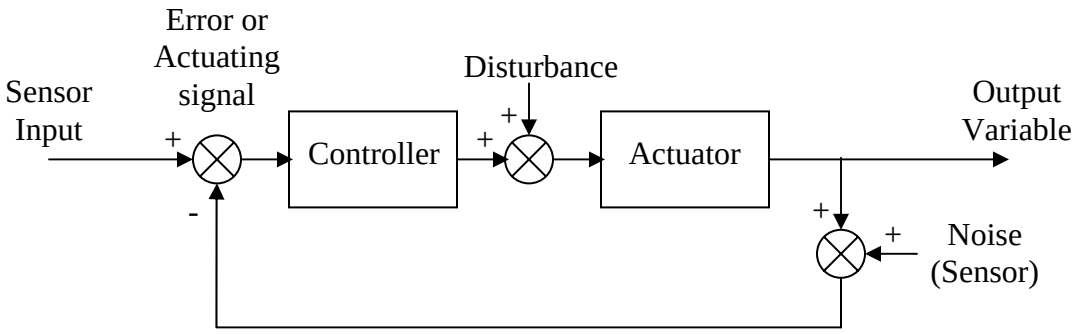


Figure 1.1. Block Diagram representation of a closed-loop control system.

The control techniques under investigation are reactive controller, linear estimator, and artificial neural network estimator. In reactive controller, the pursuing robot (predator) will intermittently initiate image capture from the camera sensor at regular time-steps to obtain information from the field. The controller will calculate the amount of reorientation and forward movement that the predator needs to perform in a straight direction towards the prey. Reactive controller is the most simplistic among the three and it does not require memory capability of the states of the predator or prey in the previous time-steps. The linear estimator is more advanced than the reactive controller as it takes the previous states as inputs to the control algorithm. Based on the current and previous states, the controller estimates the velocity of the moving object and its position in the next time-step. Instead of pursuing the object in a straight, direct path, linear

estimator allows the robot to intercept the object in the next time-step. The following figure illustrates the pursuit algorithm behind the linear controller.

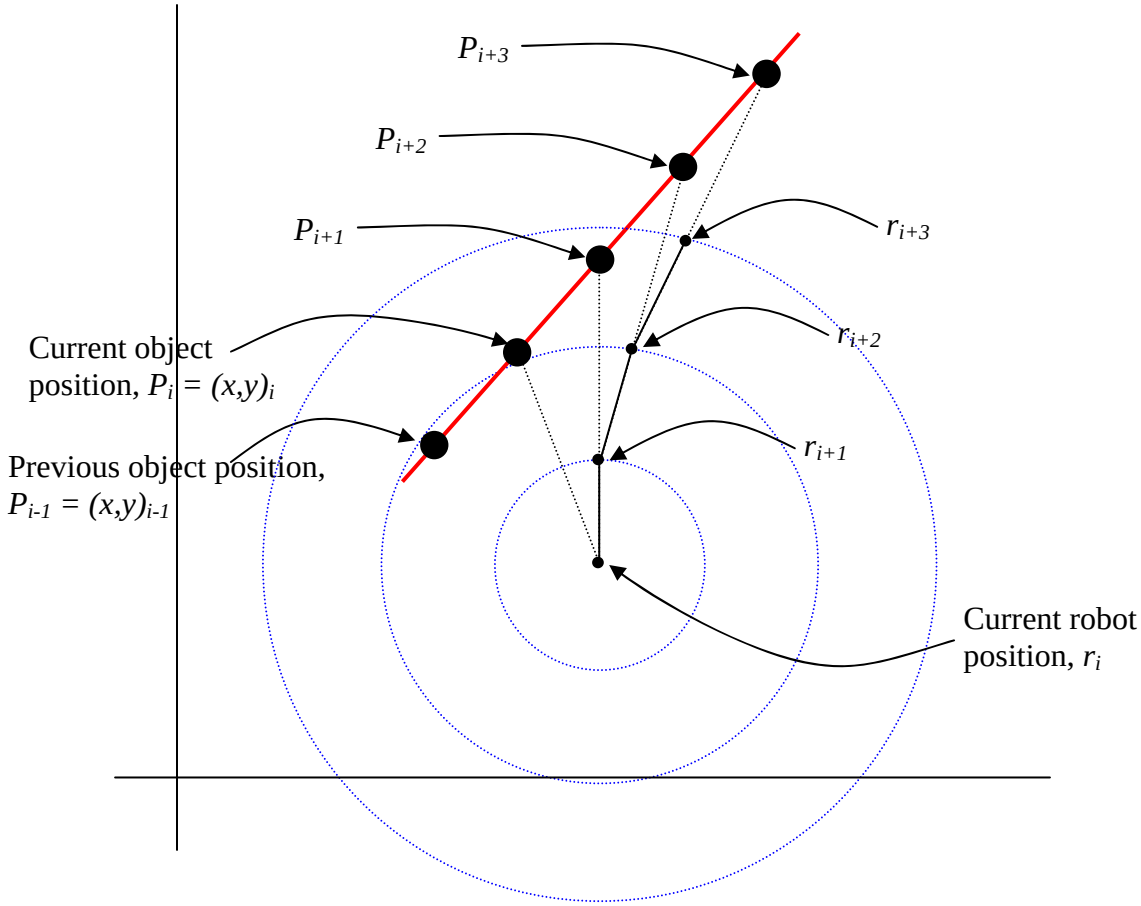


Figure 1.2. Illustration of pursuit algorithm of an estimator controller.

The third controller incorporates artificial neural network (ANN) to estimate the positional state of the prey in the next few time-steps. The ANN is modeled upon human brain's interconnected network of neurons and their processing [7,8]. In order to use this

control algorithm, the ANN must first be trained on a set of examples. During the training period, the robot will observe and learn nonlinear movement patterns of the prey and adjust its parameters to minimize the estimation error cost function. Upon completion of training, it will be able estimate the position of the prey in the next several time-steps with significant accuracy using its learned knowledge.

Selection of most suitable controller is done using several criteria, including comparison of intercept time, steady-state positional error, i.e. final distance between prey and predator at steady-state in the event that interception is not accomplished, and steady-state time (settling time), i.e. the time it takes to reach steady-state. Computational time is also of interest as it will introduce time delay which can be considered as a disturbance to the system. Lastly, the controllers are also compared based on their complexity, including the level of difficulty involved in understanding the algorithm, programming the controller and implementation in the experiments.

The objectives of this study are:

1. To explore three control techniques -- reactive controller, linear estimator, and artificial neural network estimator -- as they are applied in prey-predator setting of mobile robots system,
2. To simulate the controllers as implemented in the prescribed scenario in both computer and experimental environment.
3. To measure the fitness of each controller based on a set of goodness criteria.

In Chapter 2 of this thesis the details of the controllers -- algorithm architecture, mathematical derivation, and performance analysis -- are presented as they are modeled

in computer simulation. In Chapter 3, the control algorithms are integrated in a real-world experimental environment, and software and hardware requirements are discussed at great length, followed by conclusions in Chapter 4.

Chapter 2 – Control Algorithms and Computer Simulations

2.1. REACTIVE CONTROLLER

Reactive controller is possibly the most simplistic control algorithm that is applicable in this type of mobile robot tracker setting. The basic concept is as follows:

1. The prey is roaming within vision proximity of predator.
2. The predator 'sees' the prey through the camera which refreshes at predefined time-step value. At every time-step, the predator 'opens its eyes' and calculates the required degrees of rotation and translational movement that it needs to perform to reach the prey.
3. The predator rotates and moves forward within the available timeframe, before it 'opens its eyes' again and recalculates the new pursuit direction.

In essence, using reactive controller, the predator always moves towards the prey without performing estimation calculations of where the prey might be in the future. This is analogous to the behavior of animals with low-level brain power, or even in the case of an infant as he/she attempts to follow a moving object. The infant will track a moving object by rotating his/her head to keep it within his/her view window, and approaching the object in a straightforward direction [9].

The predator's starting position is first initialized at random coordinate within the window frame of the prey path. The prey is pre-programmed to follow certain paths (unknown to the predator) or move in a random manner. Currently, available path

libraries are circular path, rectangular, infinity-shaped, and random paths. The path dimension can be modified by altering the radius of circle or length and width of rectangle. Noise is also added to simulate a more realistic experimental environment with the presence of disturbance at sensors.

$$(x, y)_{i \to I, prey} = path(x, y)_{i \to I} + noise_i$$

In reactive controller, the predator 'reacts' in reflex to the positional state of prey at every time-step. The desired predator state, i.e. $(\bar{x}, \bar{y})_{i, pred}$, is the current prey state, i.e.

$$(\bar{x}, \bar{y})_{i, pred} = (x, y)_{i, prey}$$

Based on the current and desired state of predator, the errors -- positional (Δx,Δy) and angular ($\Delta\alpha$) -- can be calculated.

$$(\Delta x, \Delta y)_{i, pred} = (\bar{x}, \bar{y})_{i, pred} - (x, y)_{i, pred}$$
$$\Delta\alpha_{i, pred} = arctan(\Delta x / \Delta y)$$

The movement sequence starts with re-orientation, followed by forward move. The time required for rotation is

$$t_{i,rotn} = \Delta\alpha_{i, pred} \times \frac{rotation\_period}{2\pi}$$

where *rotation_period* was experimentally determined from the predator robot and represents the amount of time for the predator to perform a full revolution about its center axis; its magnitude is the same whether the rotation is clockwise or counter-clockwise. The remaining time available (if any) is used to move forward, which subsequently displaces the predator closer to the desired coordinate.

$$t_{i,\,forw} = time\_step - t_{i,rotn}$$
$$x_{i+1,\,pred} = x_{i,\,pred} + \left[V_{pred} \sin\left(\Delta\alpha_{(i+1)-i,\,pred} \right) \times t_{i,\,forw} \right]$$
$$y_{i+1,\,pred} = y_{i,\,pred} + \left[V_{pred} \cos\left(\Delta\alpha_{(i+1)-i,\,pred} \right) \times t_{i,\,forw} \right]$$

where V_{pred} is the forward speed of the predator robot as determined from experiment, and $\Delta\alpha_{(i+1)\text{-}i,pred}$ is the angular difference between the absolute orientation of the robot in the next and current time-step. Upon completion of the movement sequence by predator, the prey will have already advanced by a time-step and reached its next coordinate along the path line. The tracking error is calculated as

$$error_{i+1} = (x, y)_{i+1,\,prey} - (x, y)_{i+1,\,pred},$$

and logged for performance analysis of the controller. This (looping) procedure continues indefinitely until the prey is intercepted, or steady-state error is achieved, i.e.

$$(x, y)_{I,\,prey} \cong (x, y)_{I,\,pred}$$
$$or$$
$$\left| error_I - error_{I-1} \right| \le tolerance$$

The sequential steps of the reactive control algorithm are presented in the flow diagram in Fig. 2.1. With this control algorithm, we can foresee that steady-state error is likely to be the factor that terminates the mission as the predator is merely following without capability to intercept, unless is $V_{pred} >> V_{prey}$.

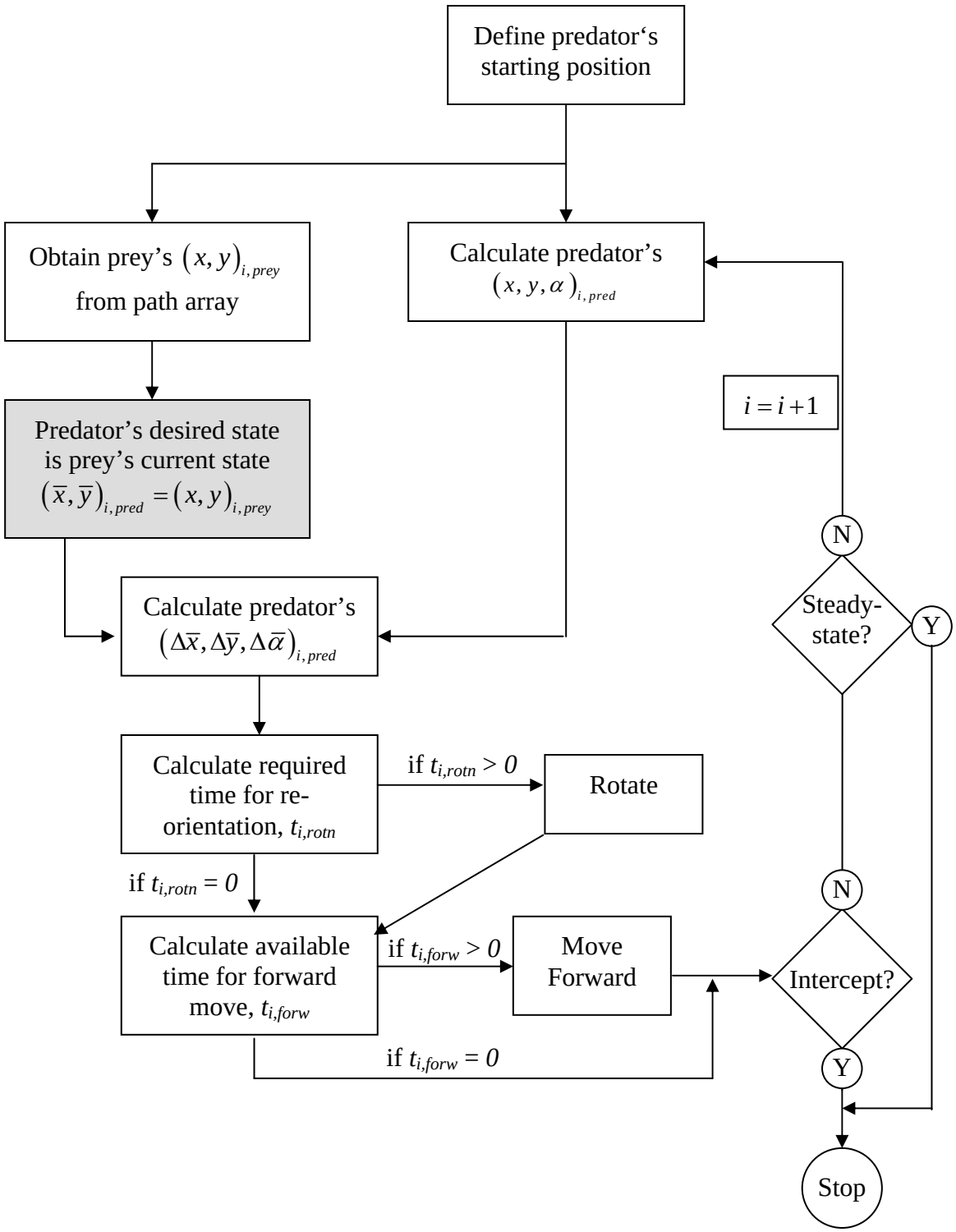


Figure 2.1. Flow chart of logic steps of a reactive controller algorithm.

2.2. LINEAR CONTROLLER

To reduce steady-state error, possibly allowing prey interception, the robot needs to have the ability to forecast the prey state in the next time-step. With a linear estimator controller, instead of pursuing the object by following, the robot estimates prey coordinate in the next time-step and attempts to intercept it.

2.2.1. Algorithm

The overall scheme of the linear estimator is very similar to the reactive controller; the prey and predator positions are first initialized, the predator's desired state is determined, error correction takes place by rotation and forward movement. The main difference is in the determination of predator's desired state. With the linear estimator, the predator observes the prey's states (positions) in the current and previous time-steps, and estimates the prey's velocity $\hat{V}_{i,prey}$ (assuming it is constant).

$$\hat{V}_{i,prey} = \frac{(x,y)_{i,prey} - (x,y)_{i-1,prey}}{time\_step}$$

The prey's coordinate in the next time-step is estimated by

$$\begin{aligned}(\hat{x},\hat{y})_{i+1,prey} &= (x,y)_{i,prey} + \hat{V}_{i,prey} \times time\_step \\ &= 2(x,y)_{i,prey} - (x,y)_{i-1,prey}\end{aligned}$$

which becomes the predator's desired state in the next time-step:

$$(\overline{x},\overline{y})_{i+1,pred} = (\hat{x},\hat{y})_{i+1,prey}$$

The algorithm then continues in the same manner as reactive controller does, exiting the closed-loop only when stopping criteria are met. The sequential steps of linear estimator controller are presented in the flow diagram in Fig. 2.2.

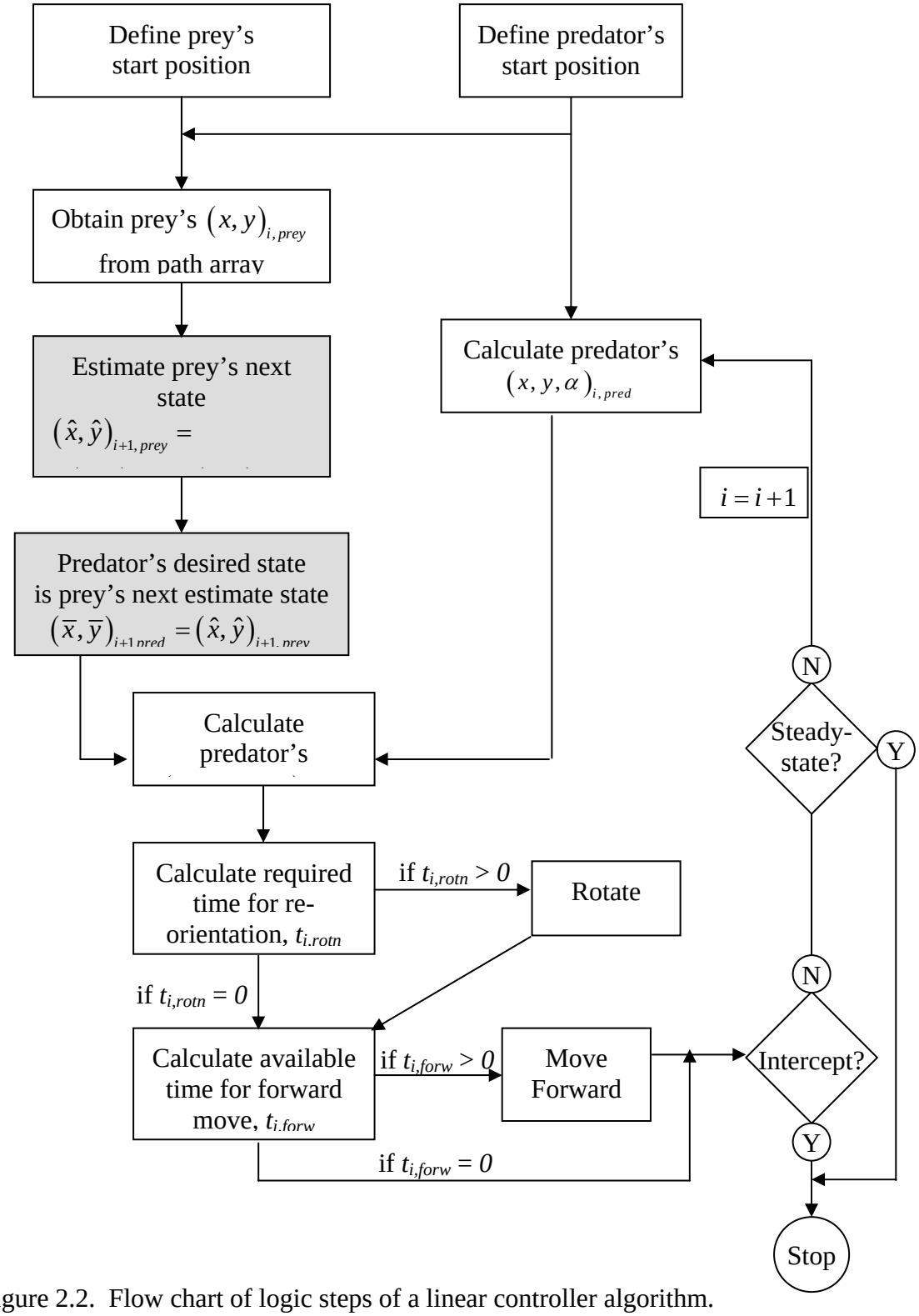


Figure 2.2. Flow chart of logic steps of a linear controller algorithm.

2.2.2. Analysis of Linear Estimator Controller

The estimator controller calculates the speed of the object and estimates its position a time-step later, with the assumption that the object's speed is constant and the path would be a line (linear path).

$$\hat{V}_{i,prey} = \hat{V}_{i+1,prey} = constant$$

However, this presents a source of error to the system, as this assumption is likely to be inaccurate. On one hand, the linear estimator pursues the object at the future time-step and expects to intercept it rather than just follow its path. On the other hand, the estimator's accuracy is affected by the lack of validity of the assumption. When the object is indeed traveling in a straight line, the linear estimator is expected to perform well. However, when it makes a sharp turn, for example at a corner of a rectangular path, linear estimator will tend to misguide the predator and cause a momentary overshoot as a result of the predator's momentum. A similar result is also expected when the prey moves randomly. To obtain an optimum advantage from linear estimator while keeping the error minimum, the time-step has to be kept relatively low. Large time-step causes the robot to move towards a desired position, estimated to be the prey's next coordinate, over a longer period while the error remains uncorrected and accumulated. Overall, the linear controller should improve the tracking performance of the reactive controller. The ability to intercept the prey's path will eventually depend on the predator's speed as well.

2.3. ARTIFICIAL NEURAL NETWORK ESTIMATOR

With the linear estimator, the controller runs into a problem from the oversimplifying assumption of the prey's path linearity resulting in inaccurate estimation. To circumvent this issue while still taking the advantage that linear estimator offers, the camera refresh rate can be increased, i.e. the time-step must be decreased, to reduce the accumulation of estimation error. In essence, this becomes an optimization problem in which the estimation accuracy and the time-step size compromise each other.

The invalid linearity assumption is a predicament that the ANN estimator can answer to, since the ANN has nonlinear estimator properties. Artificial neural networks, or more commonly referred to as neural networks, are a computer architecture systems modeled upon human brain's interconnected network of neurons [9]. It imitates the brain's ability to sort out patterns and learn from trial and error, observing and determining the relationships that exist within the presented data. Neural networks in general have two layers of neurons: hidden layers and output layers. The hidden layers' neurons have in general nonlinear and differentiable activation functions. A basic representation of ANN architecture is given in following diagram (Fig. 2.3) [10].

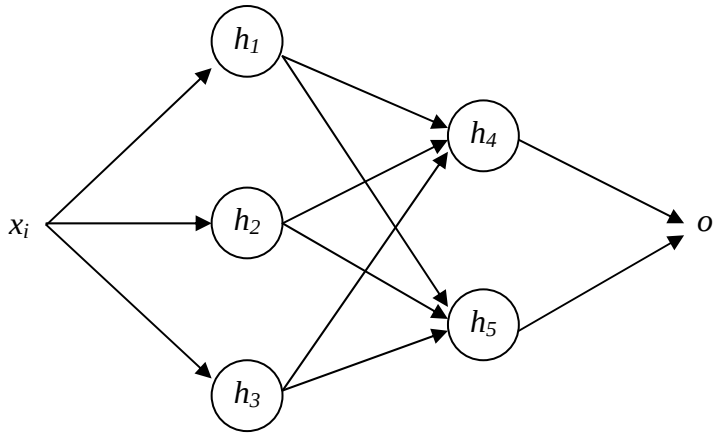


Figure 2.3. Generic architecture of artificial neural network. The network has an input x_i, two hidden layers (the 1st hidden layer with nodes h_1, h_2, h_3, and the second with nodes h_4 and h_5), and output o.

One way of interpreting this architecture is as follows: input data x is transformed into a 3-dimensional state by vector $h_{1\text{-}3}$ which in turns is transformed into a 2-dimensional state by vector $h_{4\text{-}5}$, which is finally summed to generate the predicted output o. This forward calculation through the network is called *feedforward*.

2.3.1. Algorithm

There are three different types of learning model for neural network: reinforcement learning, unsupervised learning, and supervised learning [10,11]. In reinforcement learning, the network is not presented with data input. There is a "judge" that knows the cost function, but the network does not. The judge/critic gives feedback in the form of reinforcement (value of the cost function, or good/bad, but no gradient information is provided) [11]. The inputs are generated as the system interacts with the

environment. For every time-step i, the environment induces input x_i to the system, which in turn generates an output y_i. The output is calculated by solving an optimization problem which minimizes the cost function E. The network is "rewarded" with positive or negative reinforcement signal to help determine the direction of learning in the subsequent time-steps. As learning continues, a more solid relationship will be established between x and y.

In unsupervised learning, input x is given and cost function E is determined from *a priori* assumptions. However, the output targets are unknown. One way of solving unsupervised learning is by time-series prediction where the known input data are fed forward through the network to predict the outputs. After some time delay, more data are obtained and used to calculate and "back-propagate" the error.

In this study, the learning method used is supervised learning, where the input X, the target Y, and the cost function E to be minimized are given to the network. With this learning model, the implementation of neural network algorithm will involve training, or learning, and simulation phases. During the training phase, the network is presented with a set of example pairs -- inputs and targets --, and its task is to generate inference of the mapping of the example pairs. Having established the mapping relationship, the network is then able to estimate the outputs of another given set of inputs with similar behavior. In mobile robot tracking setting, the predator will be trained with numerous cycles of a given path of the prey. It will observe and identify the "regular path", and during simulation, the predator will be able to calculate the estimated position of the prey in the next time-step. The possibility of learning is probably the most interesting aspect of

neural network that often makes it more favorable than others, for example the reactive and linear controllers used in this study.

2.3.2. ANN Training

The class of neural networks explored in this study is the multi-layer perceptron (MLP), where the computation is done is two steps: feed-forward and error back-propagation (EBP) [8]. In the feed-forward step, inputs are passed through the network and multiplied with the weights coefficients connecting the input layer and the hidden layer. The MLP network with EBP described by **Kecman** [8] is used as the main reference in this algorithm. The ANN architecture used in the context of this study is shown in Fig. 2.4.

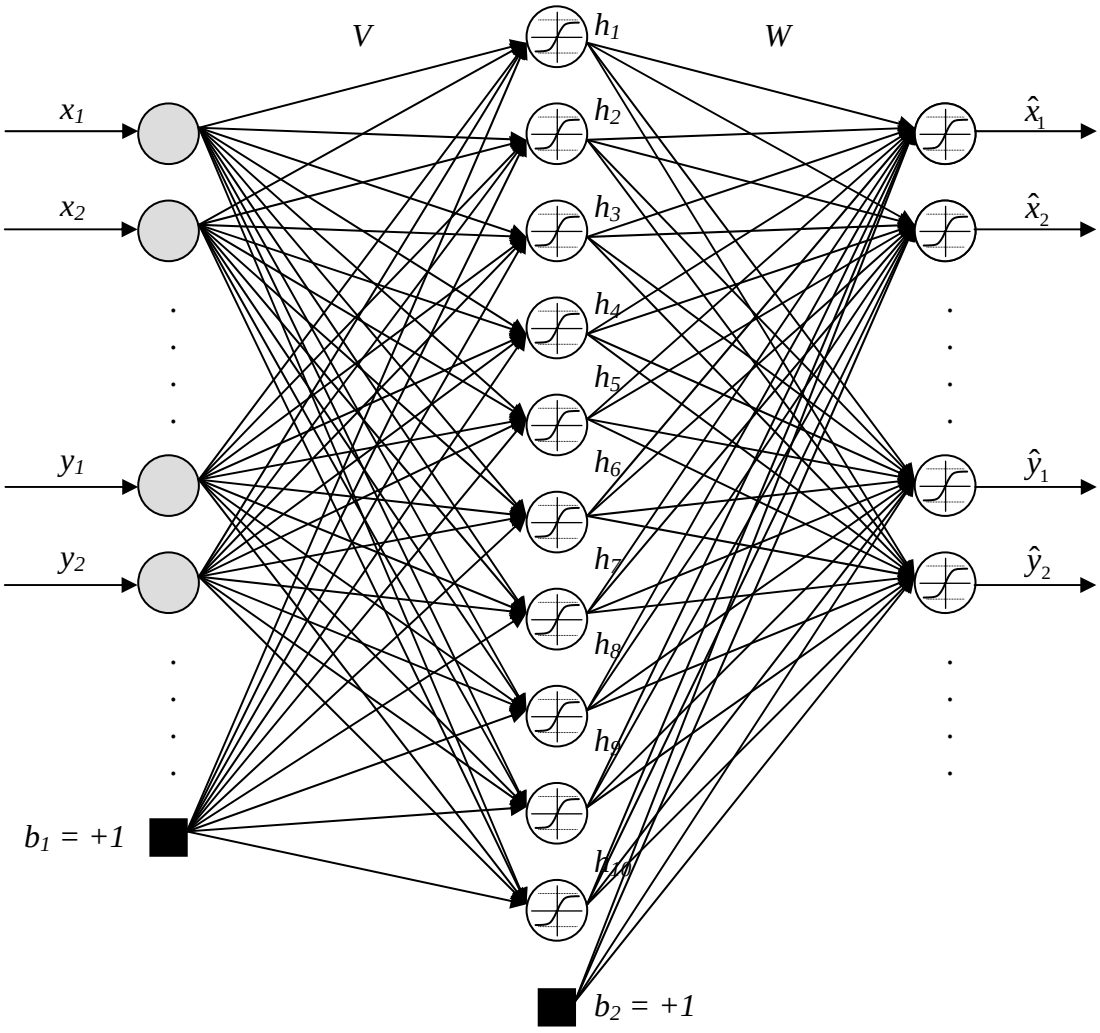


Figure 2.4. ANN architecture with Multi-Layer Perceptron (3 layers): input, hidden, and output layers. The input layer acts as a buffer (linear activation function). The hidden and output layers have sigmoidal activation functions.

The input to the network is the positional state *x* and *y* of the prey for *I* time-steps, where *I* is the number of previous pairs of prey position (*x*,*y*), including the current pair. The input is represented by the following matrix:

$$\boldsymbol{x} = \begin{bmatrix} prey(x)_{(t-I)\to t}^{T} \\ prey(y)_{(t-I)\to t}^{T} \\ b_1 \end{bmatrix}$$

where t is the current time. The size of $\boldsymbol{x}$ is N, where $N = 2I + 1$. The input signal to each hidden layer neuron is given by

$$u_k = \sum_{j=1}^{J} v_{kj} \boldsymbol{x}_j$$

Each of the hidden layers neurons is a summing junction, where the weighted input values are combined and passed to the activation function. The hidden layer is augmented with a bias term $b_2 = +1$. The activation function used in the hidden layer is sigmoidal (squashing) activation function -- in our case, the unipolar logistic function [8]. Its mathematical equation is as follows

$$h = \frac{1}{1 + e^{-u}}$$

and Fig. 2.5 shows the graph of this function:

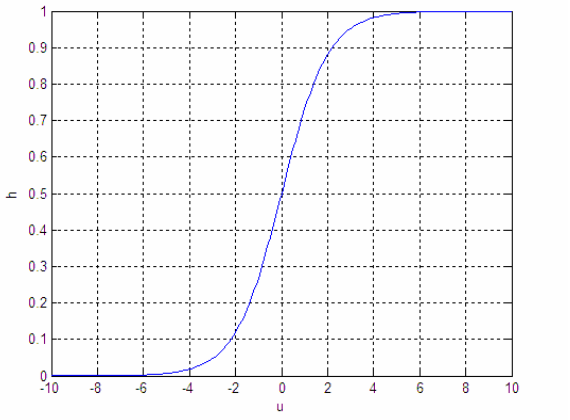


Figure 2.5. Unipolar logistic activation function.

The output layer sums the weighted values of the hidden layer output and passes the results to its activation function, which is also the unipolar logistic function. The output from the output layer matrix is as follows

$$\boldsymbol{o} = \begin{bmatrix} prey(x)^{T}_{(t+1)\to(t+K)} \\ prey(y)^{T}_{(t+1)\to(t+K)} \end{bmatrix}$$

where K is the number of coordinate pairs that are estimated by the network. The cost function to be minimized is the sum-of-error-squares of the estimated outputs $\boldsymbol{o}_{pk}$ with respect to the desired outputs $\boldsymbol{d}_{pk}$, and P is the number of data sets used for training:

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}\left(\boldsymbol{d}_{pk} - \boldsymbol{o}_{pk}\right)^2$$

2.3.3. MLP algorithm with off-line EBP

Without going through the lengthy derivations, the procedural summary of MLP network with off-line EBP calculation is detailed as follows [8]:

Initialization

Given P measured data pairs for training:

$$X = \left[\boldsymbol{x}_p, \boldsymbol{d}_p, p = 1,..., P\right]$$

with input vector and desired output

$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & \ldots & x_n & +1 \end{bmatrix}^T$$

$$\boldsymbol{d} = \begin{bmatrix} d_1 & d_2 & \ldots & d_K \end{bmatrix}^T$$

Choose learning rate η and define maximum allowable error, E_{max}

Initialize weights $\boldsymbol{V}_p\left(J-1, I\right)$ and $\boldsymbol{W}_p\left(K, J\right)$

Feed-forward

1. Calculate the hidden layers nodes states and from them the outputs of the network (output layer neurons):

$$y_{jp} = f_h\left(u_{jp}\right), \qquad o_{kp} = f_o\left(u_{kp}\right)$$

2. Calculate the sum of cost function E_p for all the data pairs:

$$E_p = \frac{1}{2}\sum_{k=1}^{K}\left(\boldsymbol{d}_{pk} - \boldsymbol{o}_{pk}\right)^2 + E_p$$

Off-line Back-propagation

3. Calculate output layer weight changes Δw_{kj} :

$$\Delta w_{kj} = \sum_{p=1}^{P}\left(\boldsymbol{d}_{pk} - \boldsymbol{o}_{pk}\right) f_{ok}^{'}\left(u_{kp}\right), \qquad k = 1,\ldots,K, \quad j = 1,\ldots,J$$

4. Using the chain-rule (EBP), calculate hidden layer weight changes Δv_{jn} :

$$\Delta v_{jn} = \sum_{p=1}^{P}\boldsymbol{x}_{np} f_{hj}^{'}\left(u_{jp}\right)\sum_{k=1}^{K}\delta_{okp} w_{kjp}, \qquad j = 1,\ldots,J-1, \quad n = 1,\ldots,N$$

Parameters Update

5. Update the output layer weights w_{kj}:

$$w_{kj} = w_{kj} + \eta\Delta w_{kj}$$

6. Update the hidden layer weights v_{jn}:

$$v_{jn} = v_{jn} + \eta \Delta v_{jn}$$

Stopping criteria

7. If $p < P$, go to step 3, otherwise proceed to step 11
8. One cycle of learning epoch is completed: $p = P$. If $Ep < Emax$, terminate network training, otherwise go to step 3 for a new cycle with incremented epoch value.

This network has been coded in Matlab and the source listing can be found in Appendix A. Prior to applying the code to the prey-predator system, it was first tested with several benchmark problems, including the XOR problem, N-parity, and sin(x)sin(y) [13]. The results from these benchmarking tests are also included in Appendix A.

2.3.4. ANN Controller

Neural network's ability to nonlinearly predict prey's position in future time-steps presents a promising improvement of performance by ANN controller compared to reactive and linear estimator. If the system's learning during training phase is well-performed and mapping inference between inputs and desired outputs is well-established (thus creating a high level of confidence on the estimator), during experiments, the robot's speed can be increased to allow much quicker response and possibly reduce the time-to-intercept.

Prior to training, several neural network parameters need to be initialized, including the learning rate η, maximum allowable error E_{max}, maximum epochs *MaxEpochs*, and the number of neurons in each layer: input *I*, hidden *H*, and output *O*. The available input array for the network contains *(x,y)* pairs of prey state from previous time-steps. However, for robustness, instead of inputting the actual x_i and y_i values, inputs to the network are modified to be the incremental change Δx and Δy values from one time-step to the next.

$$input_i = (x, y)_{i,\,prey} - (x, y)_{(i-1),\,prey}$$

Several input pairs from the current and previous time-steps are used to map several pairs in the later time-steps, as shown in the Fig. 2.6.

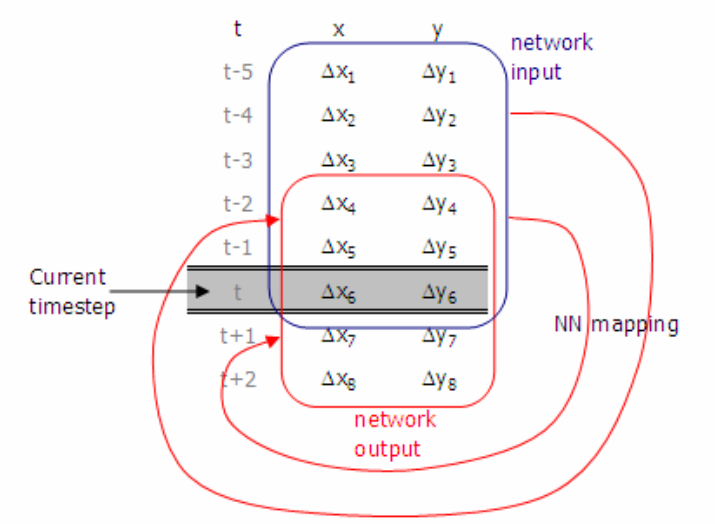


Figure 2.6. ANN mapping between input and output of a coordinate data set.

ANN training is performed following the MLP algorithm explained in section 2.3.2 to generate weight matrices *V* and *W*, which are saved for operation (simulation). With the

weight matrices, during simulation -- for example as given in Fig. 2.6 -- the network will be able to use the collected prey data up to the current time-step to estimate the output matrix:

$$input = \begin{bmatrix} \Delta x_1 & \Delta y_1 \\ \Delta x_2 & \Delta y_2 \\ \Delta x_3 & \Delta y_3 \\ \Delta x_4 & \Delta y_4 \\ \Delta x_5 & \Delta y_5 \\ \Delta x_6 & \Delta y_6 \end{bmatrix}, \quad output = \begin{bmatrix} \Delta x_4 & \Delta y_4 \\ \Delta x_5 & \Delta y_5 \\ \Delta x_6 & \Delta y_6 \\ \Delta x_7 & \Delta y_7 \\ \Delta x_8 & \Delta y_8 \end{bmatrix}.$$

Once the output matrix is generated, the prey's coordinates can be mathematically calculated. For example, prey coordinate in time-step $t = 7$ (at current time-step, $t = 6$) from Fig. 2.6 is estimated by

$$\left(\hat{x}, \hat{y}\right)_{t+1,\, prey} = \left(x, y\right)_{t,\, prey} + \left(\Delta x_t, \Delta y_t\right)$$

This then becomes the predator's desired state:

$$\left(\overline{x}, \overline{y}\right)_{i+1,\, pred} = \left(\hat{x}, \hat{y}\right)_{i+1,\, prey}$$

ANN offers a large degree of flexibility where the size of inputs and outputs are variable. The time-step at which predator coordinate is estimated is also modifiable. For example, if we desire to predict prey coordinate at $t = 8$, the estimate will be

$$\left(\hat{x}, \hat{y}\right)_{t+2,\, prey} = \left(x, y\right)_{t,\, prey} + \left(\sum_{t=7}^{8} \Delta x_t, \sum_{t=7}^{8} \Delta y_t\right)$$

Once the desired predator coordinate in the future time-step is determined, reorientation and forward move commands are executed and the looping steps are

repeated again until the stopping criteria are satisfied. The sequential steps of ANN estimator controller are presented in the flow diagram in Fig. 2.7.

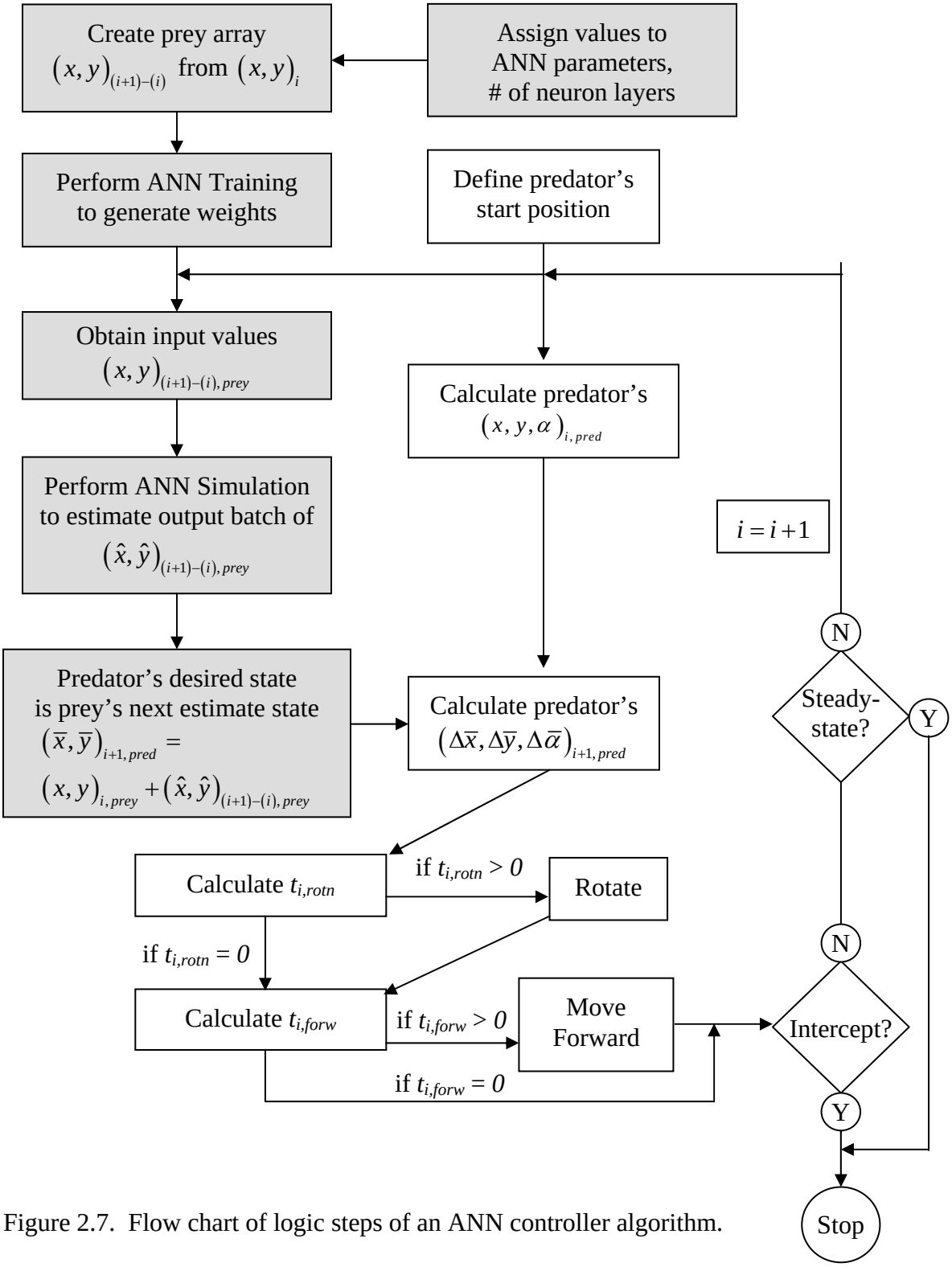


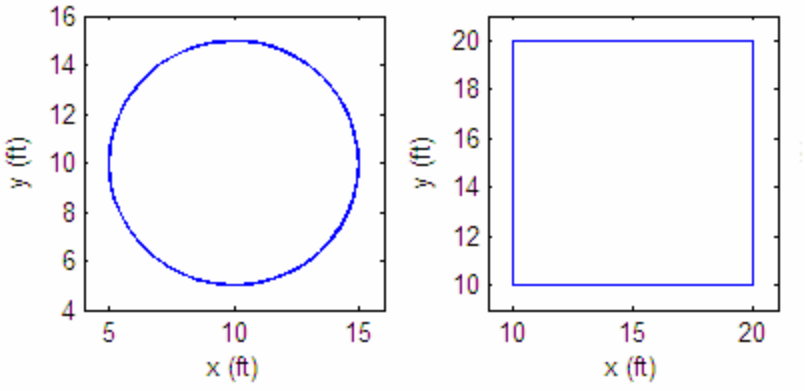
Figure 2.7. Flow chart of logic steps of an ANN controller algorithm.

2.4. MATLAB SIMULATION RESULTS

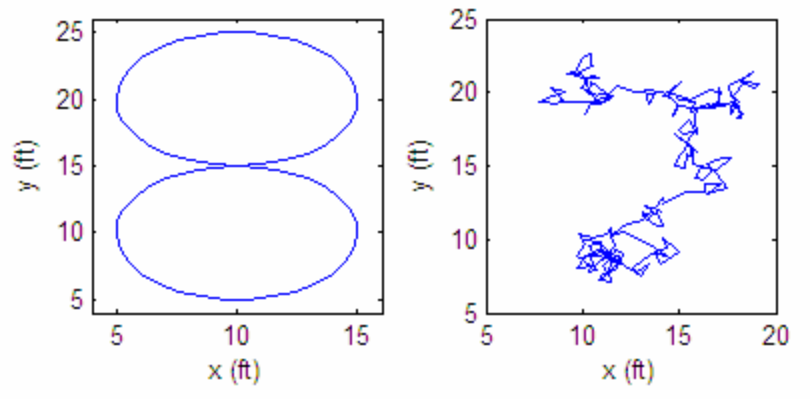
Modeling with Matlab is performed to simulate the performance of each controller in assisting the robot to pursue, and possibly intercept, moving object. The simulations were carried out with the following parameters varied:

1. Type of controller: reactive, linear, ANN controller
2. Object path pattern: circular, rectangular, infinity-shaped, random
3. Time-step: 1.3 and 2.3 seconds

The different types of controllers are discussed in Sections 2.1-3 where each algorithm is explained in details. The object moves at constant speed following several given paths: circular, rectangular, infinity-shaped, and random. The paths are as shown in Fig. 2.8 below. For meaningful comparison and simulation consistency, the distance moved by the object in every time-step has been ensured to be one unit foot (1 ft).



(a) (b)



(c) (d)

Figure 2.8. Variety of prey path patterns used for computer simulation: (a). circular, (b). rectangular, (c). infinity-shaped, and (d). random.

In the simulations, noise is added to each path so as to create a more realistic test environment, closer to what should be expected in an actual experiment.

The time-step, i.e. the time available for the predator to perform its pursuit actions -- re-orient and move forward -- before the prey takes the position given in the next array element, is a parameter that can be varied. Larger time-step value allows more time for the predator to chase after the prey. In essence, increasing time-step will have the same effect as increasing the predator's rotation and forward speed, or decreasing the prey's movement speed. Based on understanding of the controller's algorithms, it is expected that the predator will have better chance of intercepting the prey if it can move at a higher speed.

In these simulations, the rotation period (s/revolution) and horizontal speed (ft/s) of the predator are obtained from the robot through real-time measurements.

$$rot\_period = 5.9 \ \frac{seconds}{revolution}$$

$$forward\_speed = 0.4625 \ \frac{ft}{s}$$

The general characteristics of the hardware system used for the predator and prey are as follows:

- Both the predator and prey are mobile robot systems with servo-powered wheels to produce horizontal movement in the two-dimensional plane.
- The wheels are individually powered by the servos and the speed is variably adjustable, thus able to bring about nonlinear movement to the robots.
- The rotation period is 5.9 s/rev and maximum forward speed is 0.4625 ft/s for both the prey and predator.
- The sequence of events to facilitate pursuit by the predator is (a) re-orientation and (b) straightforward move.

The predator's system with an integrated controller is a sampled data system which are subject to the above capabilities and limitations. In the pursuit of the prey, the predator's system is desired to have similar speed as the object. Furthermore, the sensor devices need to have a high sampling rate, failure of which may result in misinterpretation of data. The time-step has two competing effects towards the predator's controller system: it gives the robot more time to complete its pursuit between sightings, but it also limits the number of sightings by robot. For successful experiments, the predator needs to have a sufficiently high sampling rate, enough to capture the complete dynamic of the prey, thus preventing incidents of false interpretation of prey's position.

The results presented in the following section compare the performance of each controller when run at two different time-step values: 1.3[s] and 2.3[s]. In the first case, when time-step is 1.3[s], the maximum distance that can be traveled by the predator is when it is moving in a straightforward line without requiring reorientation of its direction.

$$\begin{aligned} distance_{1.3s} &= V \times timestep \\ &= 0.4625\left(\frac{ft}{s}\right) \times 1.3(s) \\ &= 0.60 \quad ft \end{aligned}$$

In the same manner, when the time-step is 2.3[s], the maximum distance traveled is 1.06 ft. It has been mentioned earlier that the prey's speed is always held constant at 1 ft/time-step in all circumstances. The following sections show how speed is a significant factor for the predator to be able to intercept the prey. When the predator is moving too slow, regardless of controller type, it will never be able to capture the prey (when the prey keeps moving away from the predator), although a better controller will be able to reduce the steady-state error.

2.4.1. Results for Reactive Controller with Varied Time-step

The following figures (Fig. 2.9-10) present the results -- pursued path mapping and error plot -- of the predator as it attempts to intercept a prey moving in an infinity-shaped path. Time-step values used are 1.3[s] and 2.3[s]. Green circles indicate position of the prey, while red asterisks for the predator. The blue solid straight lines show the distance between predator and prey at any different time-steps.

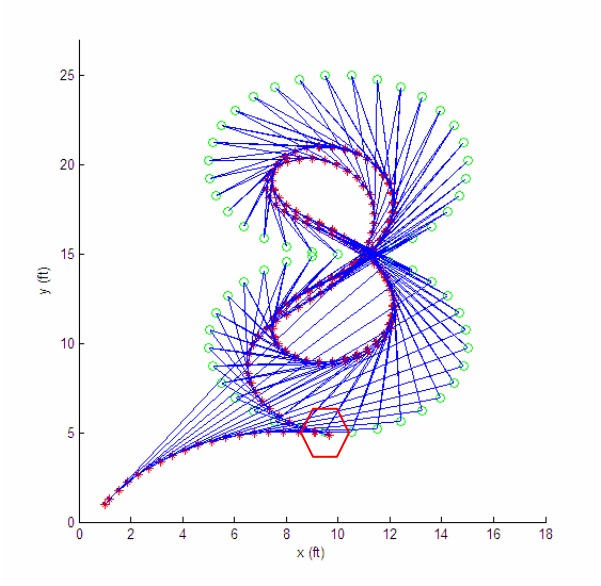


Figure 2.9. Position plot of reactive controller with time-step 1.3[s]. Note: green circles indicate prey's position at any given instant, while red asterisks indicate the predator's position. The blue solid straight lines show the distance between predator and prey at any time-steps. Marked with the hexagon is when the prey literally "runs into" the predator (given the prey's direction).

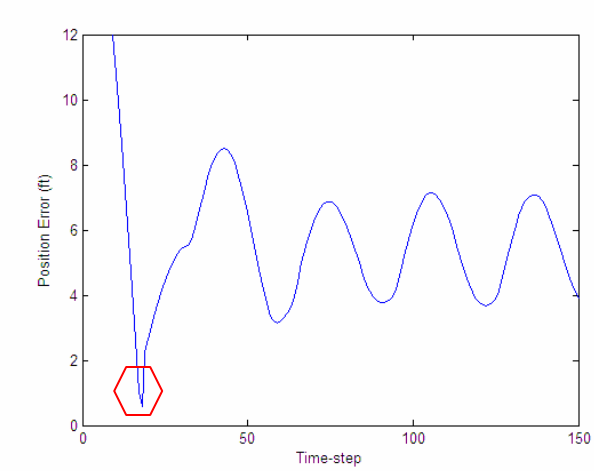


Figure 2.10. Error plot of reactive controller with time-step 1.3[s]. As in Fig. 2.9, the red hexagon shows where the prey runs into the predator.

Using reactive controller to maneuver the robot with time-step 1.3[s] proves to be insufficient for prey interception. From Fig. 2.9, it can be seen that the predator lags behind, thus it always stays inside the prey's path, never able to move quickly enough to capture the prey. Given the prey's path (clockwise at the bottom of the infinite-loop, the prey literally "runs into" the predator (around 18 seconds). Fig. 2.10 shows the error to be about ½', the smallest during the simulation. In the 52nd time-step, the predator has reached steady-state positional error of 5.47 ft as seen in Fig. 2.10. This error fluctuates from 3.71 to 7.08. This performance, however, can be improved by increasing the time-step, which in effect is the same as increasing the speed of predator, and the results are in Fig. 2.11 & 2.12.

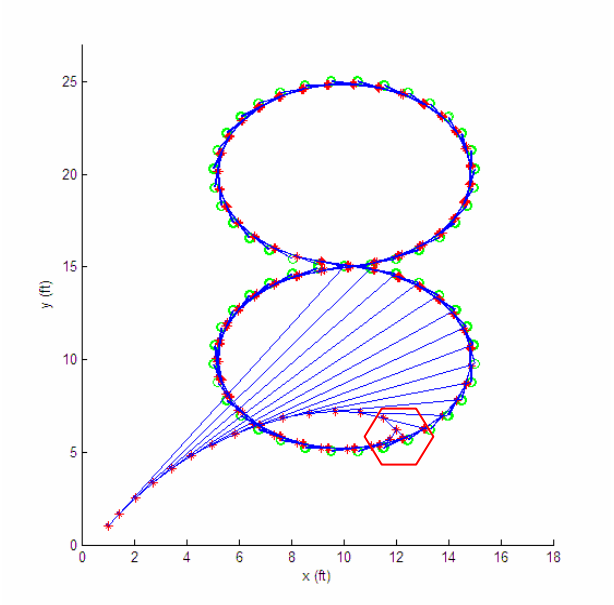


Figure 2.11. Position plot of reactive controller with time-step 2.3[s]. The hexagon marks the area where the prey almost runs into the predator (given the prey's path and predator initial position). In this case, it "drives-by". The slower time-step allows the predator to get inside the path "faster" as compared with the previous results.

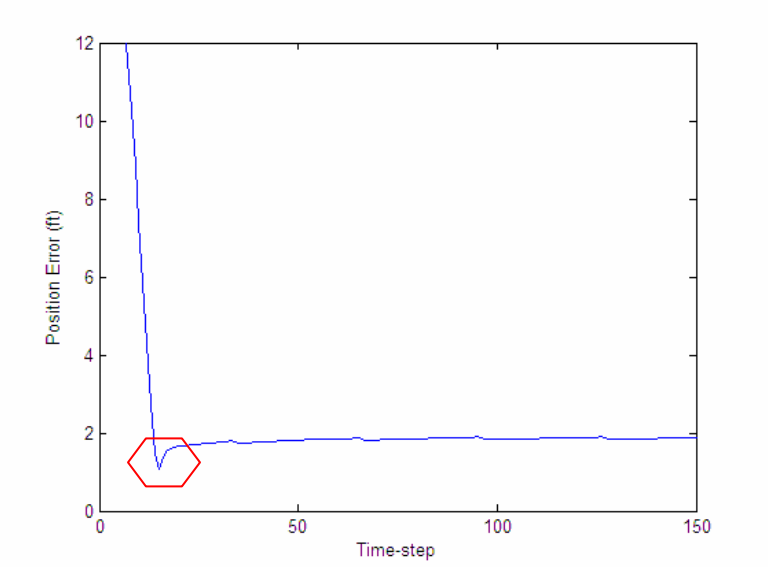


Figure 2.12. Error plot of reactive controller with time-step 2.3[s]. As in Fig. 2.11, the red hexagon shows where the prey almost runs into the predator.

Improvement in performance can clearly be seen from both figures. The resultant path of the predator resembles that of the prey very closely (Fig. 2.11). The error plot indicates that the steady-state distance error between the predator and prey is 1.71 ft, and this is achieved in the 29th time-step, much quicker than the previous case. This is evidently a significant improvement compared to when the time-step is 1.3[s], although interception still does not occur.

2.4.2. Results for Linear Controller with varied Time-step

The following figures (Fig. 2.13-14) present the results -- pursued path mapping and error plot -- of the predator as it attempts to intercept a prey moving in rectangular path. Time-step values used are 1.3[s] and 2.3[s].

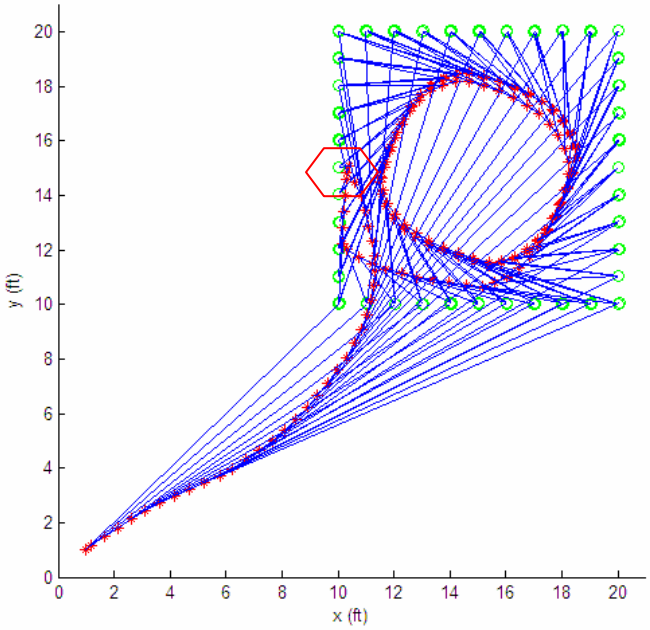


Figure 2.13. Position plot of linear controller with time-step 1.3[s]. Marked with the red hexagon is where the prey almost "runs into" the predator (given the prey's path and predator initial position).

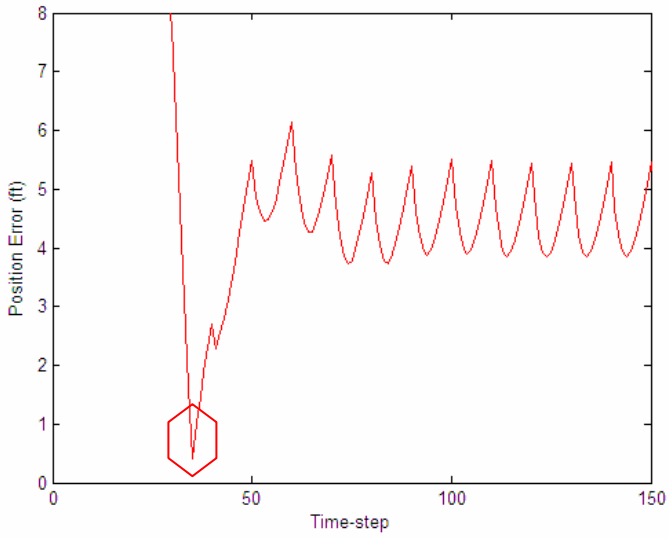


Figure 2.14. Error plot of linear controller with time-step 1.3[s]. As in Fig. 2.13, the red hexagon shows where the prey almost runs into the predator. Although the error is very small at this point, it is considered as a "fluke" case due to the prey's and predator's initial position.

Using linear estimator controller to move the robot with time-step 1.3[s] also proves to be insufficient for prey interception. Fig. 2.13 shows that the predator lags behind the prey, with the predator always being inside the prey's rectangular path. In the 74^{th} time-step, the predator has reached steady-state positional error of 4.54 ft as seen in Fig. 2.14. This error fluctuates from 3.98 to 5.15. Fig. 2.15 and 2.16 illustrate how the performance can be improved by increasing the time-step, which in effect is the same as increasing the speed of predator.

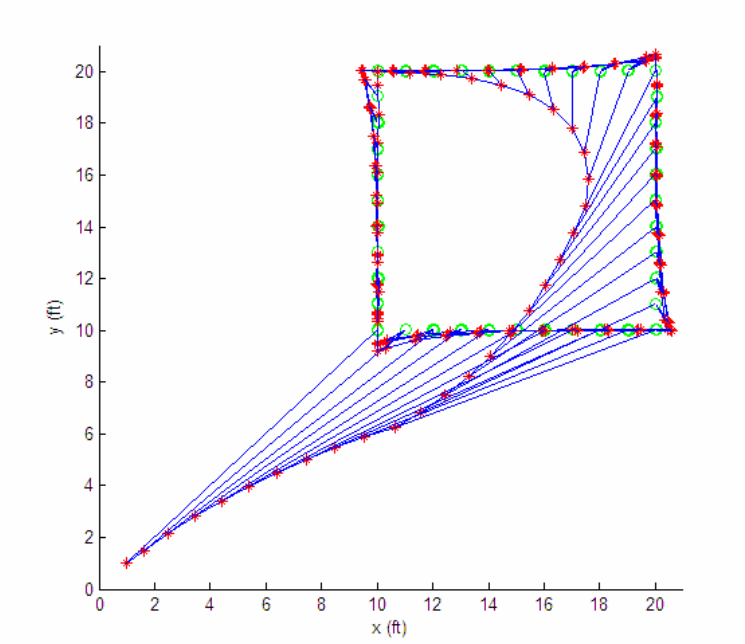


Figure 2.15. Position plot of linear controller with time-step 2.3[s].

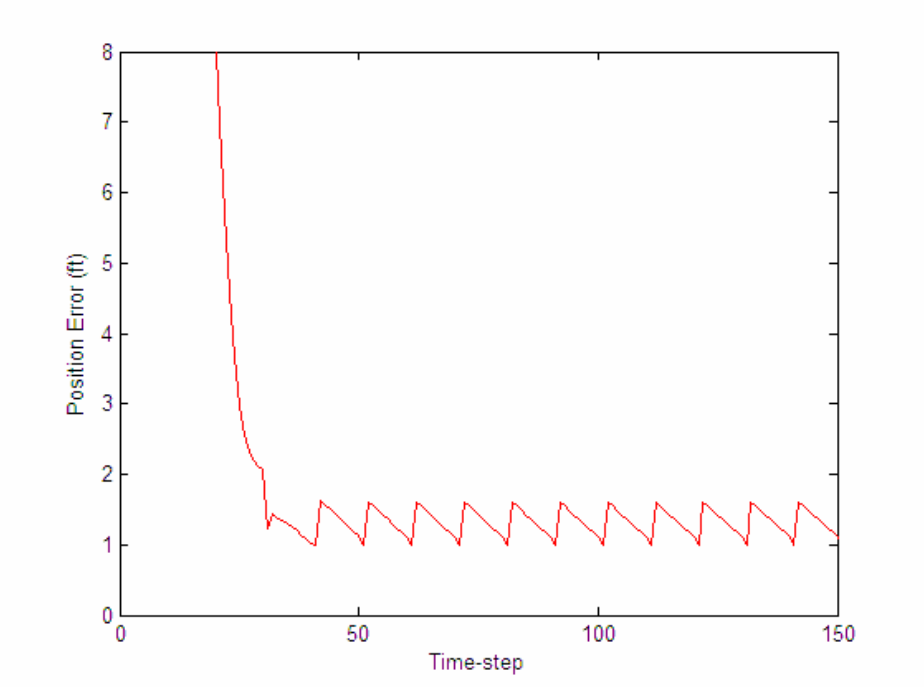


Figure 2.16. Error plot of linear controller with time-step 2.3[s].

The resultant path of the predator resembles that of the prey very closely (Fig. 2.15), i.e. rectangular in this example. The error plot indicates that the steady-state distance error between the predator and prey is 1.21 ft, and this is achieved in the 47th time-step, much quicker than the previous case. An interesting observation that one can make from the predator-prey path plot in Fig. 2.15 is how the corners of the rectangle is curved outward. This is inline with the discussion in Section 2.2.2, which states that "at a corner of a rectangular path, linear estimator will tend to misguide the predator and cause a momentary error peak as a result of the predator's momentum".

2.4.3. Results for ANN Controller with varied Time-step

The following figures (Fig. 2.17 & 2.18) present the path results and error plots of the predator as it pursues a prey moving in circular path. Time-step values used are 1.3[s] and 2.3[s].

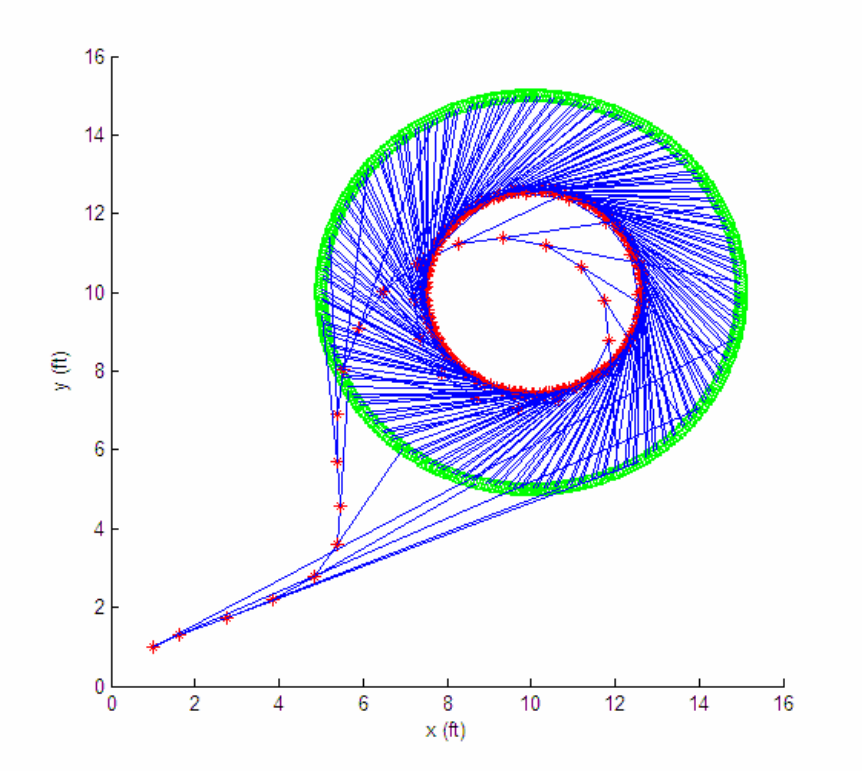


Figure 2.17. Position plot of ANN controller with time-step 1.3[s].

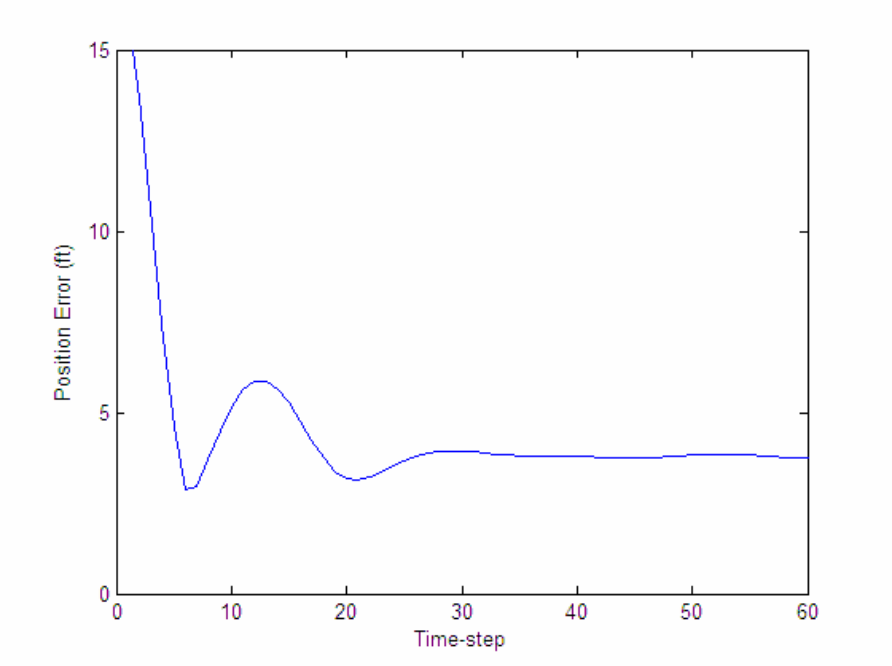


Figure 2.18. Error plot of ANN controller with time-step 1.3[s].

Using ANN estimator controller to guide the robot with time-step 1.3[s] also does not result in prey interception. Fig. 2.17 shows that the predator lags behind the prey, with the predator's path being inside the prey's circular path. The key characteristic of ANN estimator can be observed from the predator path, where it is moving in the direction towards prey position in the next several time-steps. However, with time-step setting being 1.3[s], predator's speed has a greater effect in preventing prey interception. In the 32nd time-step, the predator reaches steady-state positional error of 3.87 ft as seen in Fig. 2.18. When predator's speed is no longer a factor, as in the case when time-step is increased to 2.3[s], the performance is significantly improved as observed in Fig. 2.19 and 2.20.

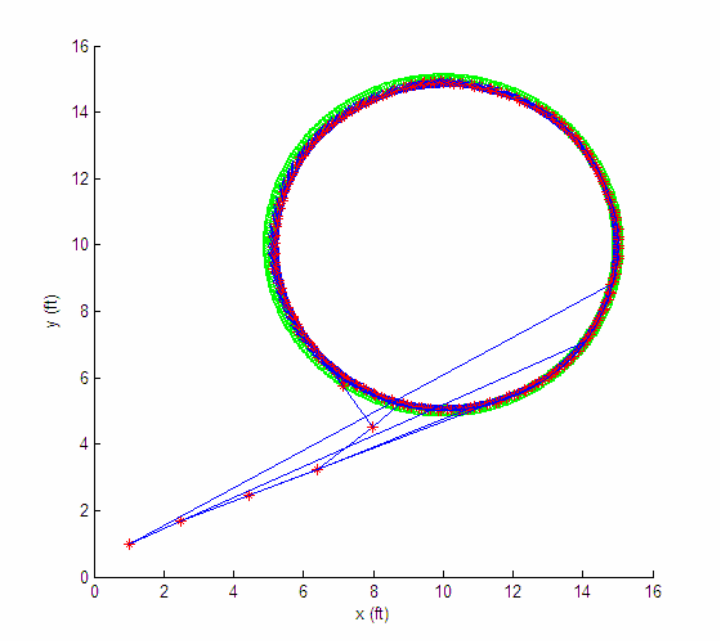


Figure 2.19. Position plot of ANN controller with time-step 2.3[s].

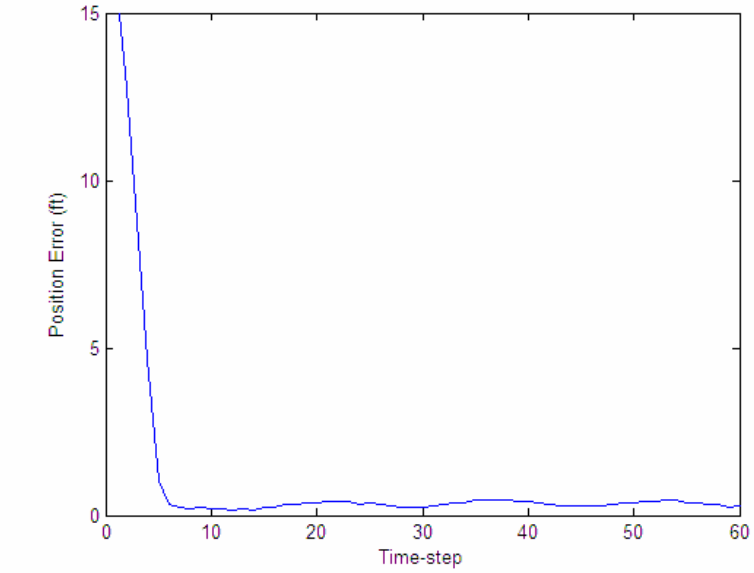


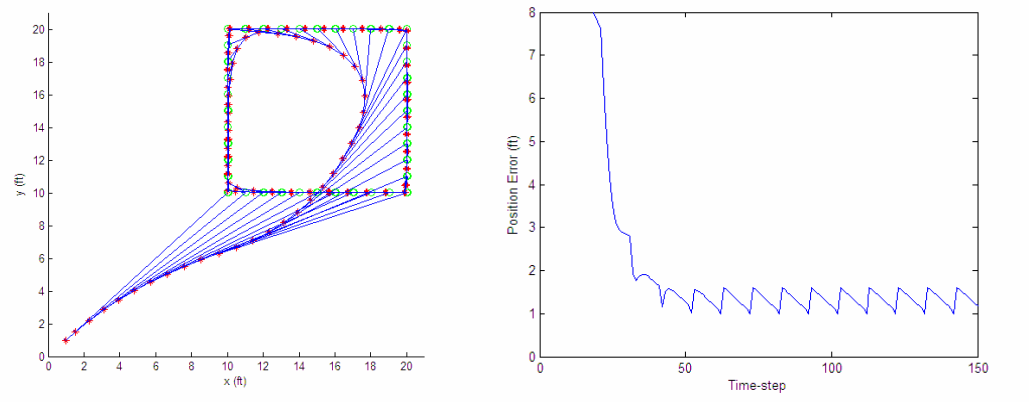
Figure 2.20. Error plot of ANN controller with time-step 2.3[s].

Improvement in performance is clearly observable from both figures; the resultant path of the predator resembles that of the prey very closely (Fig. 2.19), and the error plot indicates that the steady-state distance error between the predator and prey is 0.31 ft, achieved in 8 time-steps. In the first few time-steps, the prey almost "runs into" the predator in the first few time-steps. The pursuit direction of the predator is almost head-on with the prey's oncoming path, resulting in a sharp error reduction. The predator is able to maintain the close proximity by trailing just inches behind the prey.

2.4.4. Comparison of Controllers Performance with Varied Paths

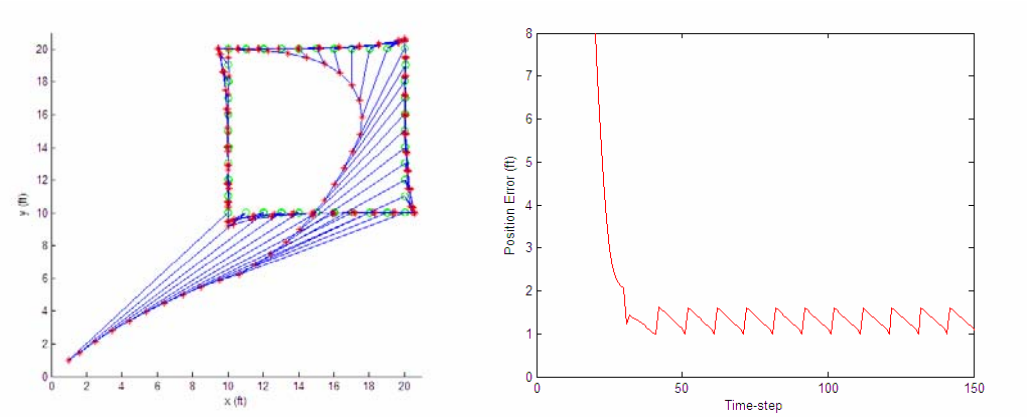
The previous section explores the effect of varying time-step on the performance of robot controller. Association has been developed between improved performance, i.e. reduced steady-state time and steady-state error, and larger time-step size, i.e. higher robot velocity. From previous results, it was apparent that ANN controller is the best controller as it reduces the system's steady-state error to the smallest value, relative to the other two controllers. However, prey's path was not held constant from one controller to another. In this section, the same path will be used to analyze the different controllers, and in the end, the collected results will be tabulated and graphed to present the overall performance.

The following figures (2.21-23) present the error plot and path of the prey and predator with time-step size of 2.3[s], as the prey moves in continuous rectangular pattern.



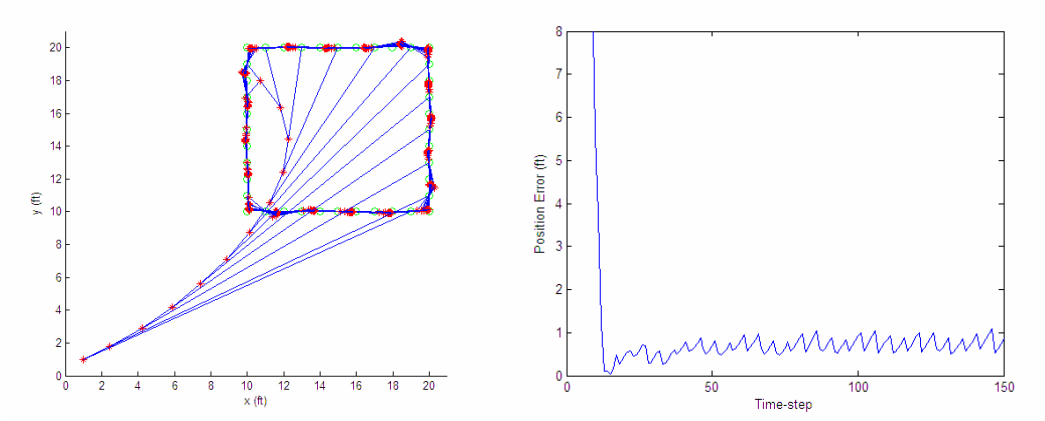
(a) (b)

Figure 2.21. Path and error plots of reactive controller with time-step 2.3[s].



(a) (b)

Figure 2.22. Path and error plots of linear controller with time-step 2.3[s].



(a) (b)

Figure 2.23. Path and error plots of ANN controller with time-step 2.3[s].

The error plots indicate substantial reduction in steady-state error when ANN controller is used, instead of reactive or linear estimator controller. At the 14th time-step, the error is minimally reduced to 0.01 ft, as seen from Fig. 2.2.3-(b). However, this is regarded as a coincidence, as the very low error is not due to the controller itself, but the initial positions and directions of the prey and predator that cause them to almost "run-into" each other. The error soon increases and reaches steady-state at the 51st time-step with average error value being 0.80 ft, fluctuating between 0.62 and 0.98 ft.

Fig. 2.23-(a) highlights one main feature of ANN estimator where it is able to predict the behavior of the prey around corners and accurately guide the robot to make the turns. ANN is well-suited to solve nonlinear problems such as in this case, and it is much better than linear estimator which assumes the prey velocity to be constant, thus resulting in corner errors as seen in Fig. 2.22-(a). The advantage of ANN estimator is also observed when simulated with other prey paths.

The following plots (Fig. 2.24-26) illustrate the comparison of steady-state errors attained by the controllers when the object is following specific paths.

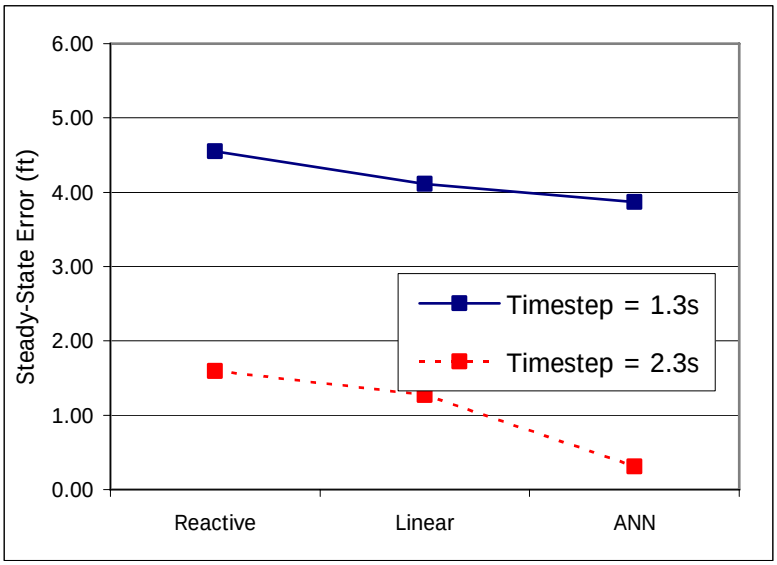


Figure 2.24. Steady-state error comparison of controllers for a circular path.

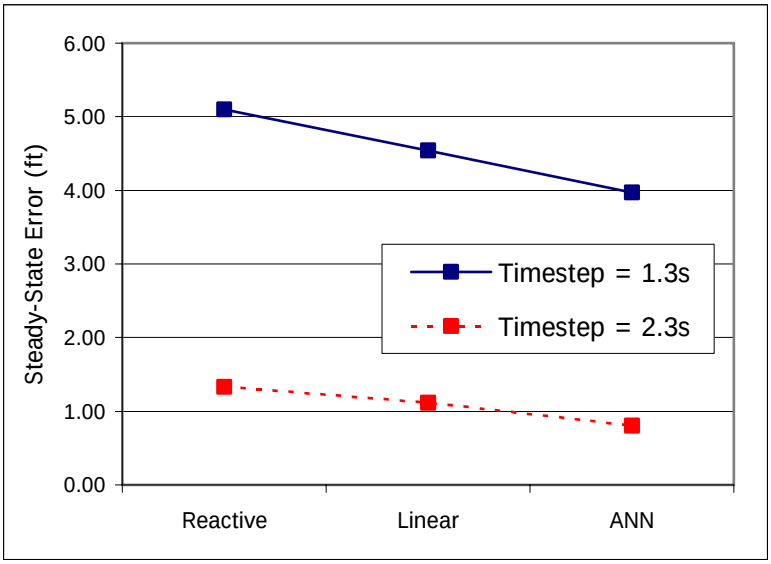


Figure 2.25. Steady-state error comparison of controllers for a rectangular path.

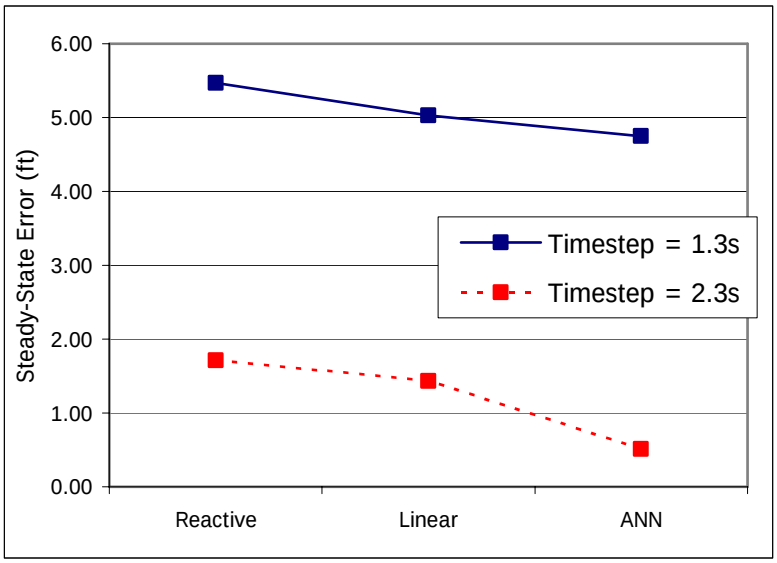


Figure 2.26. Steady-state error comparison of controllers for an infinity-shaped path.

From the steady-state plots of circular, rectangular, and infinity-shaped paths, it can be deduced that the ANN estimator is consistently better than the reactive controller and linear estimator in all the different simulation settings. ANN controller's better performance relative to other controllers is a result of its ability to accurately predict nonlinear behavior of the object. Furthermore, its estimation precision is not limited to just one time-step ahead, but, for all theoretical purposes, is boundless. Linear estimator is also better than reactive controller in all settings because of its ability to predict object's coordinate in the next time-step, although somewhat inaccurately if compared to ANN's estimation. The advantage of larger time-step which is concluded in the beginning of Section 2.4 is also shown in the plots. The red lines, which correspond to

simulations with time-step 2.3[s], are always below the blue lines where time-step is 1.3[s].

The advantage of estimator as controllers is only pronounced when the object follows a certain behavior, linear or nonlinear. As such, estimator controller is not a suitable tool to assist to robot when the object is moving randomly. The following graph (Fig. 2.27) shows result inconsistency of steady-state error values of the three different controllers when the robot is tracking a random path; no controller is specifically better than others.

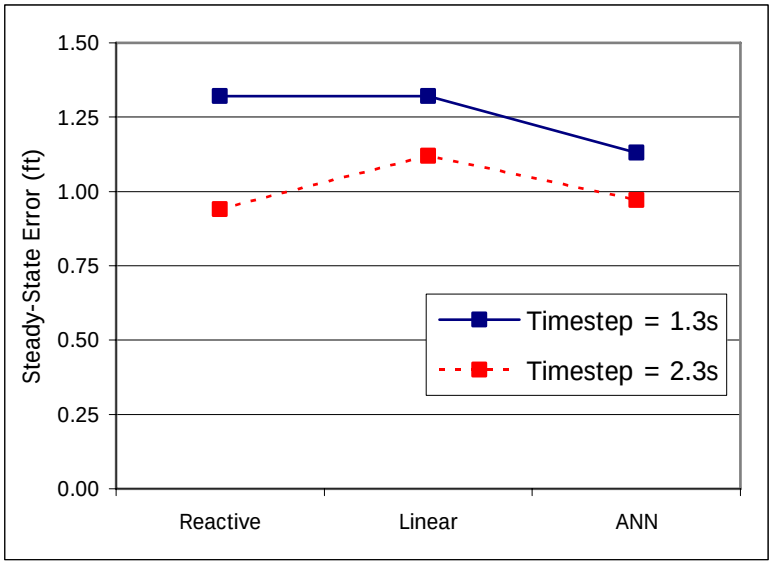


Figure 2.27. Steady-state error comparison of controllers following a random path.

Considering other fitness criteria, such as ease of programming and implementation, reactive controller is likely to be the most suitable controller to be used

to track a random object. The breakdown of ANN controller is due to its inability to learn the behavior of the object, which is expected, as the path is indeed random.

Chapter 3 – Implementation in Experimental Environments

To recreate the experiments from computer simulation setting to the real environmental setting, several aspects need to be considered, including sensor input acquisition, input processing, control algorithm, output data transmission (truncation) and command execution time. In the previous chapter, simulation with Matlab was simplified in a black-box sense, where the control algorithm takes in predefined inputs, process the data, and execute the command mathematically (Figure 3.1).

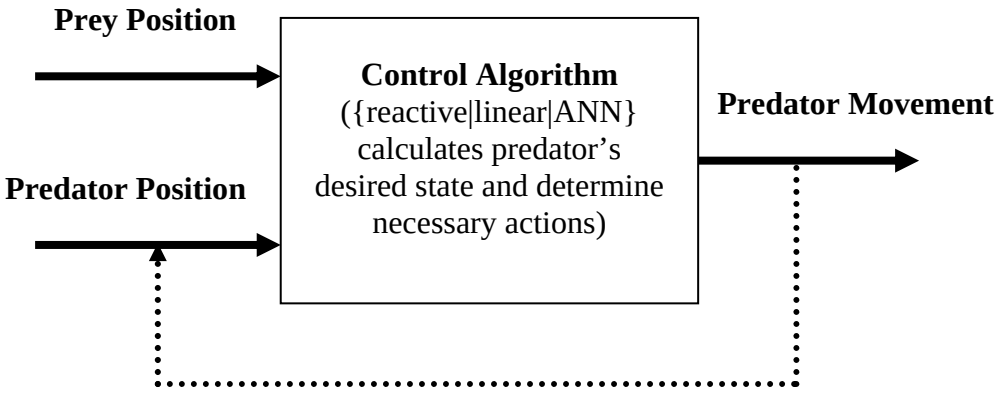


Figure 3.1. Black-box representation of inputs and output to the control system used in this study. In the computer simulation setting, the inputs are the prey's position from the path array and predator's position from the control system's output. The predator's output is calculated by $[(x,y)_{i+1} = (x,y)_i + V_i * time\_step]$. In the real-world experimental environment, the inputs are the predator's position (from overhead camera) and the prey's position (from overhead and/or on-board cameras), while the output is command data sent to the robot to perform error correction actions.

3.1. EXPERIMENT OVERVIEW

In the real experiment, obtaining the prey position input, or rather prey's relative position from the predator, is a much more complicated task. It entails image acquisition from the camera using suitable hardware, image processing to extract essential data, and data transfer to the platform that manages the control algorithm. The control algorithm structure is very similar to the one used in the Matlab simulation, detailed in sections 2.1-3. To enable predator movement, output data from the controller is transferred to the predator's hardware for interpretation and for the servos to be actuated. The schematic below (Figure 3.2) summarizes the experimental setup in a concise manner.

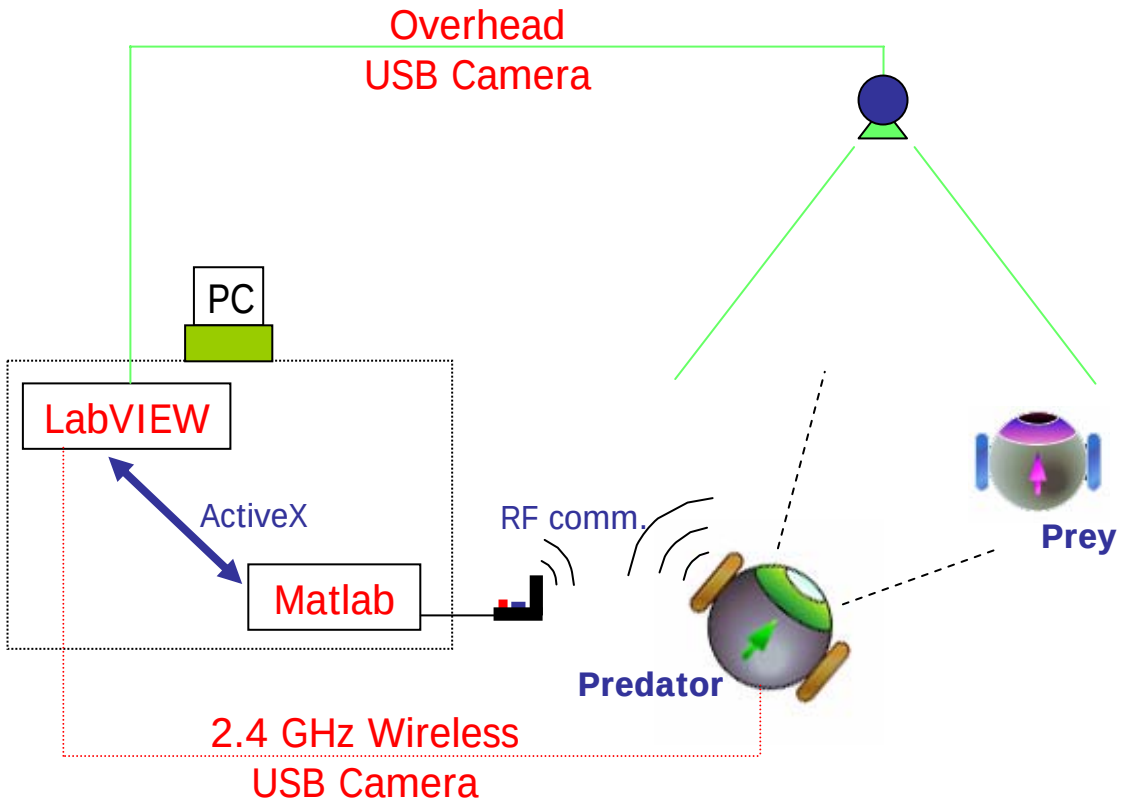


Figure 3.2. Schematic of hardware and software experimental setup in the laboratory. Vision of the field (containing the prey and/or predator) is obtained via overhead or on-board camera (wireless). Image data are transmitted back to the PC and processed by LabVIEW. LabVIEW extracts necessary information and passes the data to Matlab via ActiveX server application. Matlab performs control algorithm calculation and sends instruction via wireless RF communication to the predator's servos to move accordingly.

The predator's primary inputs from the environment are visual images obtained from cameras (see Fig. 3.2). Through experiments, the predator's performance is compared when on-board or overhead camera is used as the vision sensor. In the application, the use of overhead camera will resemble a setting where the predator has GPS capability or is able to communicate with satellite to track the prey's position. This is also in close

resemblance with the Matlab simulation. The use of the on-board camera mimics the situation in the real application where the predator may be equipped with sensors, such as IR sensor, heat sensor, or camera to track the prey's position.

The visual data are acquired by data acquisition software, LabVIEW [14], and processed by image filtering LabVIEW-add-on software, IMAQ Vision [15], to determine the positional state of prey. By communicating via ActiveX, LabVIEW passes the state information to Matlab, a numerical computing environment and programming language created by Mathworks, which serves as the main platform for data processing in the control system [16]. Matlab determines the desired predator's position in the next time-step and determines the action needed by predator. An action command is sent to robot's servos via RF signal and captured by the RF receiver on the robot.

3.2. IMAGE PROCESSING AND DATA ANALYSIS AND INTERPRETATION

To accurately determine the relative position of prey from predator in real-time and have the data ready for input to the control system, the following needs to be accomplished:

1. Image acquisition: capture continuous streams of images using compatible cameras.
2. Image processing: filter raw images, cancel noise, and extract useful information.
3. Data transfer: communicate data streams from image acquisition device to image processing software and to the control system software.

4. The system must be calibrated with the robot scale and the accuracy determined.

3.2.1. Image Acquisition

Vision of the field is obtained via cameras; there are two alternatives for the camera positioning during the experiment: overhead and on-board. The overhead camera is mounted above ground and it provides an overall view of the robot's field (playground). The on-board camera is mounted on the predator robot and it acquires visuals in front itself. In real applications of autonomous robot, the system may receive input from sensors attached to the robot (IR sensor, heat sensor, light detector) or from independent sensor units (external speed tracker, GPS system).

The overhead camera, installed 10 ft above ground, connects to the PC through the Universal Serial Bus (USB) port. The USB port provides serial bus standard for devices connection from one to another. The overhead camera used in the experiment is the *Creative NX Pro webcam* (Fig. 3.3), and its specifications are listed in Table 3.1 [17].

Figure 3.3. *Creative NX Pro webcam* that is used for overhead camera.

Table 3.1. Technical specifications of overhead camera.

Video resolution: 640 x 480 VGA
Video format: RGB-24
PC interface: USB port
Frame rate: 30 fps
Image filtering: automatic & manual exposure control and color balance
Field-of-view: 40 degrees +/- 5%

The camera was selected for its sufficiently high video resolution and suitable video format, necessary to ensure fine image quality captured by the camera. High frame rate of 30 fps is also needed to reduce time delay and ensure real-time data processing. Automatic exposure control and color balance option allows less filtering to be done by the image processing software, thus reducing computation time. Lastly, 40^{o} field-of-view allows optimal capture of the 5 x 7 ft field below the camera.

The on-board camera presents certain challenging requirements for hardware selection as it requires wireless functionality. The main challenge was to interface the camera with the image processing software -- LabVIEW and IMAQ Vision -- for real-time analysis. LabVIEW and IMAQ Vision are software products by National Instruments [14,15], and special drivers are required for the wireless cameras to be recognized by these software.

The device used as on-board camera is a surveillance-type camera, *Q-See QSWMC 2.4 GHz Wireless Camera* (see Fig. 3.4), which is DirectShow compatible [18].

This surveillance camera receives power either from 9V battery or power outlet. It is small and compact enough (1” x 1” x 0.75”) to be mounted on the robot without adding substantial weight (47g). The maximum resolution is 640 x 480 and the camera is also capable of performing raw image processing, such as exposure control and color balance.

Figure 3.4. *Q-See QSWMC 2.4 GHz Wireless Camera* used as on-board camera.

A 4-channel receiver captures the transmitted data from the camera and outputs the image in Composite Video format (NSTC) [19], which is typical format for television input. *DVD Maker USB2.0 Capture Box* by Kworld converts the Composite Video data to MPEG format and connects to the PC through USB port [20].

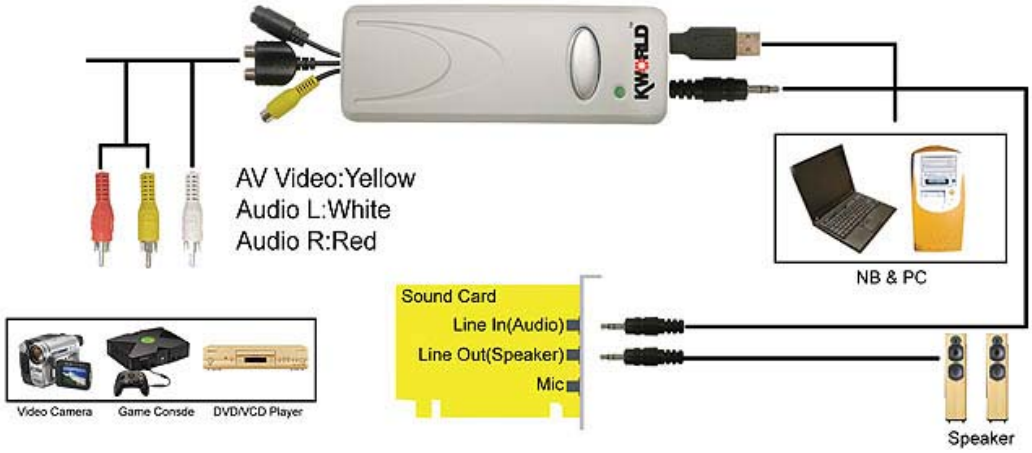


Figure 3.5. *DVD Maker USB2.0 Capture Box* schematic showing the inputs and outputs of the adapter and interfacing capability with other devices.

3.2.2. Communication between Camera and LabVIEW

Each image in the streams captured by the camera device undergoes a series of image processing steps within LabVIEW. To pass the images from the camera to LabVIEW, *NI-IMAQ for USB Cameras* add-on software is used. This software provides the ability to use USB cameras that have DirectShow filters with the LabVIEW and IMAQ Vision [21]. The cameras may operate at various resolutions and frame rates, depending on camera capabilities, and NI-IMAQ for USB Cameras will acquire and set properties using the camera manufacturer driver and DirectShow functions. NI-IMAQ for USB Cameras supports the following application development environments for PC (Windows 2000/XP):

- LabVIEW 7.0 or later with IMAQ Vision 7.1 or later
- NI Vision Assistant 7.1 or later.

NI Vision Assistant, another software product by National Instruments, is a configurable, interactive prototyping application, which allows for machine vision software development without programming [22]. Figure 3.6 describes the architecture of application of *NI-IMAQ for USB Cameras* as it relates to the camera, LabVIEW, IMAQ Vision, and Vision Assistant.

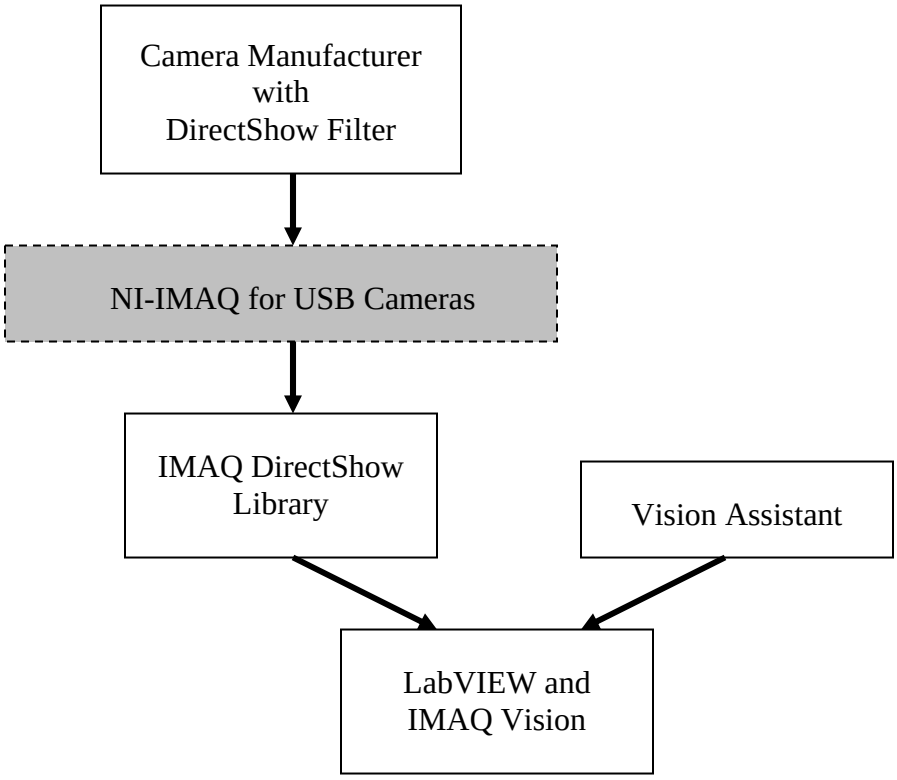


Figure 3.6. Architecture of applications for image acquisition using camera and add-on software for processing by LabVIEW and IMAQ Vision.

3.2.3. Image Processing Algorithm

NI Vision Assistant software is used to create a LabVIEW-compatible script containing sequential steps to filter noise from the image and eventually determine

object's position with desired accuracy. The main difficulty encountered while programming the LabVIEW algorithm with Vision Assistant was the enormous number of variables that have to be taken into account. Many factors affect image quality and the robot's ability to distinguish objects in an image. Figure 3.7 shows a snapshot of the original image as captured by the camera.

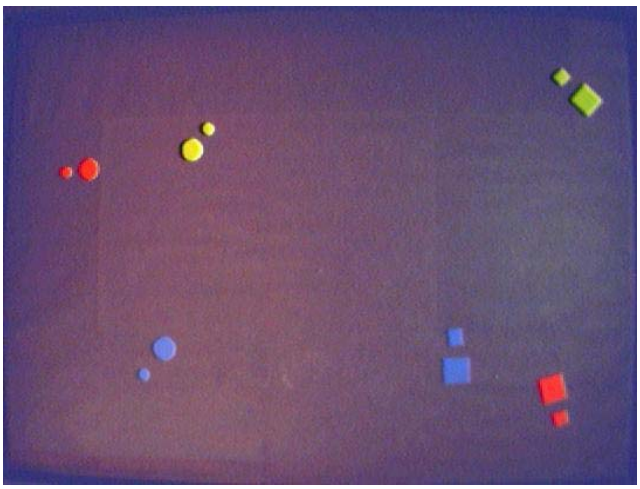


Figure 3.7. Snapshot of an original, untouched image, showing four different markers (paired shapes of different sizes) with different colors.

The primary image operation is exposure control which applies brightness, contrast, and gamma correction to each color plane separately to remove initial noise in the raw image. Other important functionalities of IMAQ Vision used in this experiment include the followings:

1. *IMAQ Color Subtract*: subtracts a color constant from an image
2. *IMAQ Color Threshold*: applies a threshold to the three planes of an RGB image and places the result into an 8-bit image

3. *IMAQ Remove Particle*: eliminates or keeps particles in an 8-bit binary image resistant to a specified number of erosions. The particles that are kept are exactly the same shape as those found in the original source image.
4. *IMAQ Convex Hull*: draws the convex hull for each particle in the image
5. *IMAQ Particle Analysis*: returns the number of particles detected in a binary image and a 2D array of requested measurements about the particle.

Figure 3.8 below shows the transformation sequence of the raw image in Figure 3.7 to processed images as the image undergoes filtering steps by IMAQ Vision.

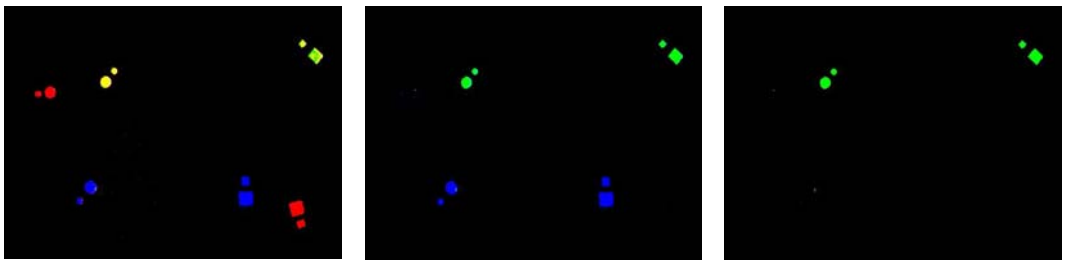


Figure 3.8. Filtration steps to remove noise and isolate green objects in the image. Shown from left to right are: original image after noise removal; red objects filtered out; final frame, blue objects filtered out (only green left).

Although Vision Assistant provides tremendous assistance in the programming of image processing steps, extensive experiments are still required to produce a robust and fault-proof image filtering application that would work in varying environmental conditions. It is worth-mentioning that at the preliminary stage of research, other approaches of image processing methods were also pursued. One successful approach was to use binary B/W camera, instead of RGB webcam, where the object was

recognized by associating its identity with its pixel area. The use of RGB webcam, however, increases the number of observable robots by at least threefold, since it allows pure color (RGB) identification.

3.2.4. Communication between LabVIEW and Matlab

After each completion of image processing LabVIEW, *IMAQ Particle Analysis* outputs measurement values of requested parameters about the object, such as *object pixel area*, *object dimension* (width and height), and *object coordinate* within the image. These parameter values are essential inputs to the control algorithm which runs in Matlab application. For Matlab to receive these data from LabVIEW, ActiveX technology needs to be incorporated. ActiveX is a distributed object system and protocol used to manage compound documents and data transfer between application [23], which is accomplished via *Object Linking and Embedding* (OLE).

LabVIEW software has integrated *ActiveX Automation* functionality that allows other programs to use and control the LabVIEW VI. Commands and data can be sent to different applications in a single format by means of invoking and getting or setting properties. Matlab can access LabVIEW VI through its *ActiveX Server*, whereby an *Application Object* exports the properties of LabVIEW [23]. Matlab command used to initialize this is

```
lvserv=actxserver('LabVIEW.Application');
```

From this Application Object, a VI Object is refered to export the properties and methods of a VI:

```
vi=invoke(lvserv,'GetViReference','file_destn\filename.vi');
```

Matlab also has direct control to remotely instruct the specified LabViEW VI to be opened and run:

```
vi.FPWinOpen=1;
vi.Run([true]);
```

The most important Matlab command line is the `invoke` command to acquire data input from LabVIEW VI of a specific parameter in LabVIEW.

```
variable = invoke(vi, 'GetControlValue','parameter_in_VI');
```

3.2.5. Data Interpretation for Coordinate Determination

3.2.5.1. Using Overhead Camera

The image data from LabVIEW are generally ready for processing and manipulation in Matlab. Using *IMAQ Particle Analysis* VI, LabVIEW is capable of determining the pixel coordinate (x,y) of objects seen through the camera. When overhead camera is used, the predator is identified by placing a marker (two circular, colored papers -- red -- of different sizes) on the robot. The smaller paper is situated “in front” of the larger one, given the robot’s direction; size differentiation is necessary to allow determination of robot orientation. The prey is identified with another marker (a green circular piece of paper) on its top. Fig. 3.9 shows a snapshot of the predator and prey in the field, with their respective identification markers.

Figure 3.9. Overhead view of the playground, as seen through the overhead camera. The predator is shown with two red markers, and the prey with a green marker on its top.

From the pixel coordinate of each object outputted by LabVIEW, the real-world coordinates of prey and predator are as follows:

$$(x,y)_{prey,real} = \frac{(x,y)_{green,pixel}}{k_C}$$

$$(x,y)_{predator,real} = \frac{\frac{1}{2}\left[(x,y)_{red\_small,pixel} + (x,y)_{red\_large,pixel}\right]}{k_C}$$

where k_C is the calibration coefficient which indicates the number of pixels that corresponds to the length in foot in real-world measurement. For the overhead camera, $k_C = 85$ *pixel/ft*.

3.2.5.1. Coordinate Determination using On-Board Camera

Coordinate determination using on-board camera is accomplished by performing perspective calculation to the images viewed by the robot. The depth of an object in an image, i.e. its distance from the camera, directly relates to the apparent dimension of the object. For an ideal camera with zero distortion, object's distance will be linearly proportional to the width or height of object as seen by the camera. However, since the camera is far from ideal, **nonlinear calibration** has to be performed to determine the distance of an object from the camera. This is done by taking snapshots of the object, which is placed at a predefined distance and incrementally retreat away from the camera. The distance from the camera and pixel height value pairs are tabulated, and mathematical relationship is derived. The plot in Fig. 3.10 shows the nonlinear relationship between the object's distance and pixel height.

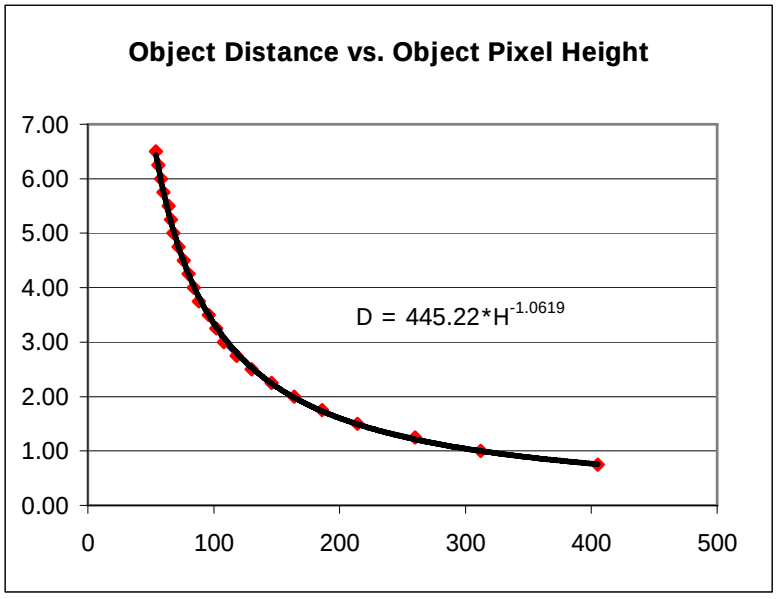


Figure 3.10. Plot of nonlinear relationship between object distance and pixel height.

This relationship is approximated by the following exponential function:

$$D = 445.22 \cdot H^{-1.0619}$$

To visualize how object distance (D) and object coordinate (x,y) relate to one another, the following diagram (Fig. 3.11) is presented to provide a better illustration.

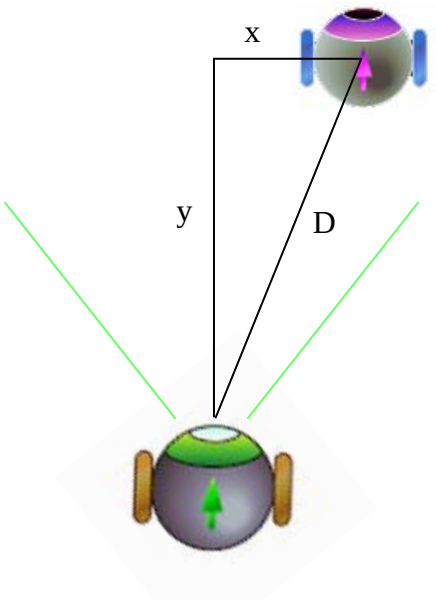


Figure 3.11. Schematic representation of relations between object's relative distance (D) and coordinate (x,y) from the robot.

The value of x_{real} can be calculated as follows:

$$x_{real} = \left(x_{pixel} - 320 \right) \times \left(\frac{H_{real}}{H_{pixel}} \right)$$

where x_{pixel} is the x-coordinate output from LabVIEW which specifies the object's horizontal distance from the left wall; *320* is half of the window width size in pixel. H_{pixel} is LabVIEW output and H_{real} is object's real height in foot length. Once x_{real} is calculated, y_{real} is given by

$$y_{real} = \sqrt{\left(D^2 - x_{real}^2\right)}$$

When both the object and robot are moving, the determination procedure of prey's relative coordinates from predator becomes more complex. Coordinate reference will continuously change from time-step to time-step as the robot rotates and moves forward. Given relative coordinate set (x_1,y_1,D_1,α_1) of the object in time-step 1, upon robot's rotation by a magnitude of θ while the object remains static, the estimated relative object coordinate in time-step 2, as viewed by the robot will be

$$\hat{\alpha}_2 = \alpha_1 - \theta$$
$$\hat{D}_2 = D_1$$
$$\hat{x}_2 = \hat{D}_2 \times \sin\hat{\alpha}_2$$
$$\hat{y}_2 = \sqrt{\hat{D}_2^2 - \hat{x}_2^2}$$

After rotation, the robot will move forward by a distance y_r, and as a result, the estimated position of the robot in the coordinate frame of the camera will be:

$$\hat{x}_{2r} = \hat{x}_2$$
$$\hat{y}_{2r} = \hat{y}_2 - y_r$$

Based on the estimated position of the static object and the object coordinate from the captured image in time-step 2 (x_2,y_2), the actual distance moved by prey is

$$\Delta x_{obj,1-2} = x_2 - \hat{x}_{2r}$$
$$\Delta y_{obj,1-2} = y_2 - \hat{y}_{2r}$$

Finally, in the original coordinate frame, the corrected object coordinate is given by

$$
\begin{aligned}
x_{2,corr} &= x_1 + \Delta x_{obj,1-2} \\
&= x_1 + \left(x_2 - \hat{x}_{2r} \right) \\
&= x_1 + x_2 - D_1 \sin\left(\alpha_1 - \theta \right) \\
y_{2,corr} &= y_1 + \Delta y_{obj,1-2} \\
&= y_1 + \left(y_2 - \hat{y}_{2r} \right) \\
&= y_1 + y_2 - \left(\sqrt{D_1^2 - \left[D_1 \sin\left(\alpha_1 - \theta \right) \right]^2} - y_r \right) \\
&= y_1 + y_2 + y_r - \sqrt{D_1^2 - \left[D_1 \sin\left(\alpha_1 - \theta \right) \right]^2}
\end{aligned}
$$

where (x_1,y_1,D_1,α_1) and (x_2,y_2) are the object's relative coordinates from the robot in time-steps 1 and 2, respectively. θ and y_r are the angle of rotation and forward distance moved by the robot as part of its pursuit routine.

3.3. CONTROL ALGORITHMS IN MATLAB

3.3.1. Algorithm Description

The general algorithms and procedures coded in Matlab for the experiments are very similar to the control algorithms detailed in the previous chapter, section 2.1-3. However, in the real experiment, instead of acquiring prey and predator coordinates mathematically, these parameters are inputted from the image acquisition system. The controller determines the desired predator position, i.e. the direction of pursuit of the predator, and calculates the time needed to perform rotation and forward move. Matlab then sends commands to the robot to actuate its servos for the prescribed length of time, before looping back to trigger input data from the camera and LabVIEW.

3.3.2. Commands Signal to Actuate Robot's Servos

The commands sent to actuate the servos are transmitted wirelessly via RF signal, the details of which are discussed in the next section. In brief, the RF transmitter is connected to the PC via serial port, and Matlab directly communicates with that specific port. To initialize the port for recognition by Matlab, a parameter is assigned as object handler for the serial port using Matlab's `serial` command. The baudrate can be selected from 1200 to 115k. Subsequently, `fprintf` command is used to write a string of character and send it to the receiver on the robot's end. The analysis of this data by the robot is a topic of discussion in the next section.

3.4. ROBOT SOFTWARE AND HARDWARE SYSTEM FOR COMMAND EXECUTION

This section provides detailed descriptions of the robot software and hardware necessary to carry out the experiments. In its operation, the robot obtains certain input from the controller and processes the inputs to bring about the output.

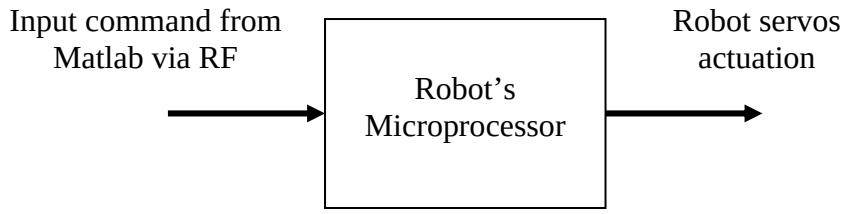


Figure 3.12. Black-box model of input-output to predator robot used in this experiment.

3.4.1. Input Commands from Matlab

One of the most important aspects of the experiment is the wireless communication between the Matlab application in the computer and the on-board program in the robot. The wireless communication device used is a set of *SureLink 900 MHz RF Module* and *QuickLink Demo Board* [24], also developed by Parallax Inc., which is compatible with the robot. Fig. 3.13 shows the SureLink Module slotted into the QuickLink Demo Board unit, which is connected to the PC via serial cable. The configuration of the wireless device is included in Appendix B.

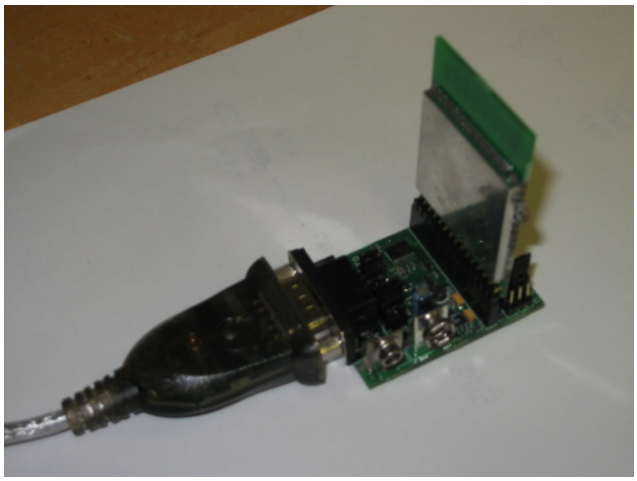


Figure 3.13. Hardware for RF communication between PC and Boe-Bot.

3.4.2. Robot Hardware and Microprocessor

The robot used in this experiment is ***Boe-Bot***, a simple mobile robot kit developed by Parallax Inc., with a BASIC Stamp microcontroller as its processing unit. The Boe-Bot is a highly customizable mobile unit that runs off of 9V battery power. The base

model comes with the *Board of Education* (BOE) Rev. C carrier board, the BASIC Stamp 2px microcontroller (see Fig. 3.14), and two servomotors. Power input into board is regulated by a three-position switch: position 0 sets the power to off, position 1 supplies power to all but the servo pins, and position 2 supplies power to the whole board. On the left edge of the board is a DB-9 serial port that allows the microcontroller to communicate with a computer. The BASIC Stamp 2px microcontroller module is an extremely fast chip that allows the Boe-Bot to perform relatively complex decision-making capabilities. The figure below shows a complete, assembled Boe-Bot kit with BASIC Stamp inset.

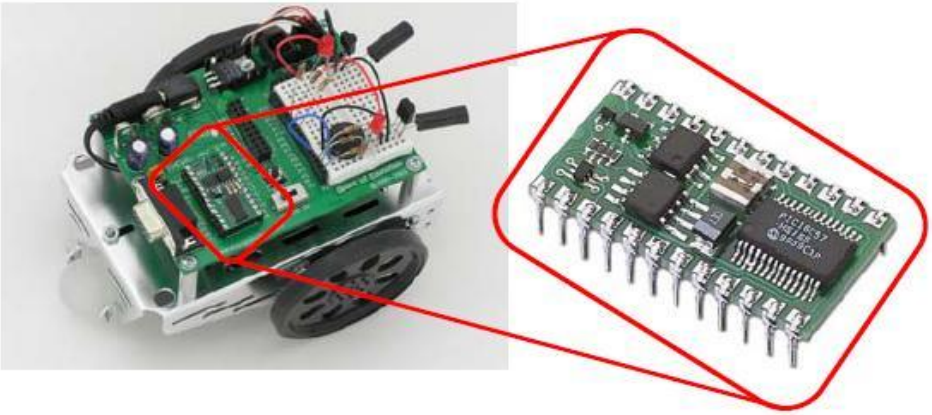


Figure 3.14. Fully assembled Boe-Bot robot with BASIC Stamp microcontroller.

The programming language used within Boe-Bot is **PBasic** [25]. It is very similar in many regards to the BASIC programming language and requires no compiler.

3.4.3. Robot Servos Actuation

The string input from Matlab is processed within PBasic to determine rotation speed of each servo. Depending upon the string characters, the servos are actuated by a certain speed in a certain direction. The servos are controlled by PBasic's `pulsout` command which delivers high signals to the servos for a specified length of time. The following command line

```
Do
   Pulsout 12, 650
   Pause 20
Loop
```

delivers a high signal to the servo assigned to Pin 12 for a time duration of 2 x 650 μs (i.e. 1.3 ms), pauses for 20 ms and repeats the loop. The resulting motion is a continuous clockwise rotation of the specified servo with full speed range of approximately 60 RPM. When coded in a `FOR` loop, simple calculation can determine the number of iterations necessary for the servo to rotate for a specific number of seconds. Fig. 3.15 illustrates servos timed motion in a timeline.

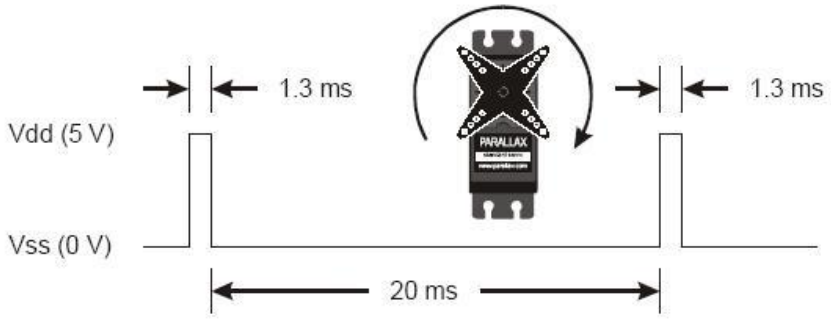


Figure 3.15. Timed motion of servo motor of the Boe-Bot [25].

The logic loop of PBasic programs for servos actuation downloaded onto the BASIC Stamp microcontroller starts with the servos being idle. Next, the microcontroller invokes for input data from the pin which is assigned to communicate with the wireless device. If no actuation command is sent by Matlab at that moment, the servos will remain idle. Otherwise, the input data will be processed to determine rotation speed and direction. The expected data is a string character input with a double-digit number which will be decomposed to two single-digit numbers. The number on the left determines the rotation of the left servo and number on the right for the right servo. The program then passes through an `IF` loop to assign time duration value for the rotation of each servo. Servos rotation will take place continuously until a new input is received by the RF receiver, where the routine above will be repeated to determine the new rotation speed and direction. The logic loop of PBasic program is illustrated in the flow diagram in Fig. 3.16.

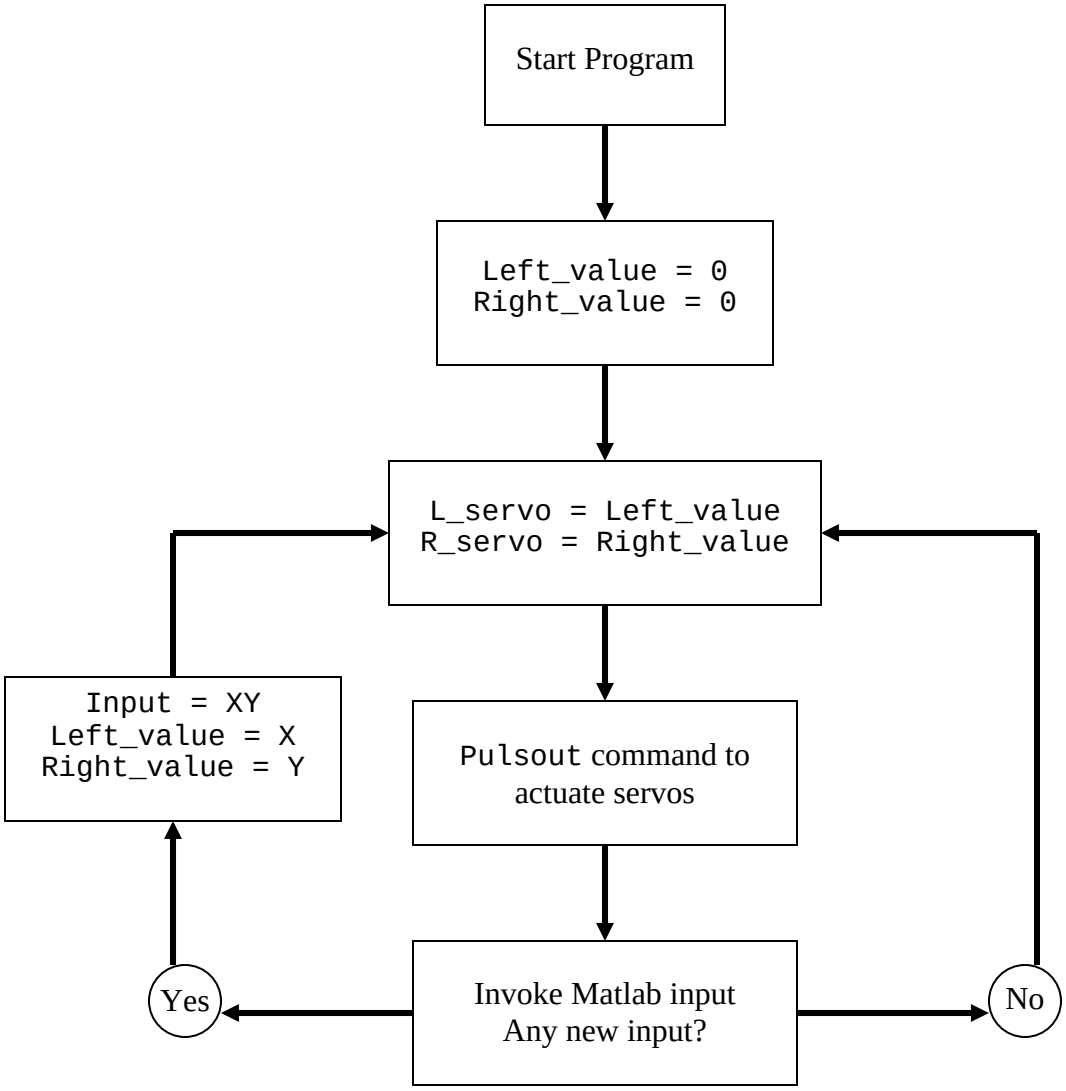


Figure 3.16. Logic loop of PBasic program downloaded onto the robot's microcontroller.

3.4.4. Methods for Time Specification of Servos Rotation

The control algorithm in Matlab determines the necessary action that the robot needs to perform in order to pursue the object. This action is defined by the rotation characteristics of the servos -- rotation direction and duration -- each performed in a

specific length of time. To control the time duration for each action, two approaches are explored, each one with its shortcomings.

The first method to control the time duration of rotation is by using `pause` command in Matlab. After the actuation command is transmitted wirelessly to Boe-Bot, Matlab application is paused for the calculated period of time, until the next command is sent again. To instruct the robot to rotate for 1s and move forward for 2s, the Matlab command lines will be

```
rotate
pause(1)
move_forward
pause(2)
stop
```

This method would be flawless and fault-proof if command data is transmitted and executed instantaneously. However, experiment showed that for every successful command sent from Matlab to the Boe-Bot, there is a time delay of approximately 0.16s. As a result, instead of rotating for 1s, the robot actually rotates for 1.16s, similarly for the forward move. Although it does not result in accumulated errors, as desired destination is re-calculated at every time-step, this overshoot cause significant error in each motion of the robot, impeding its capability to intercept the object. The diagram in Figure 3.17 shows the time delay of RF communication between Matlab and Boe-Bot.

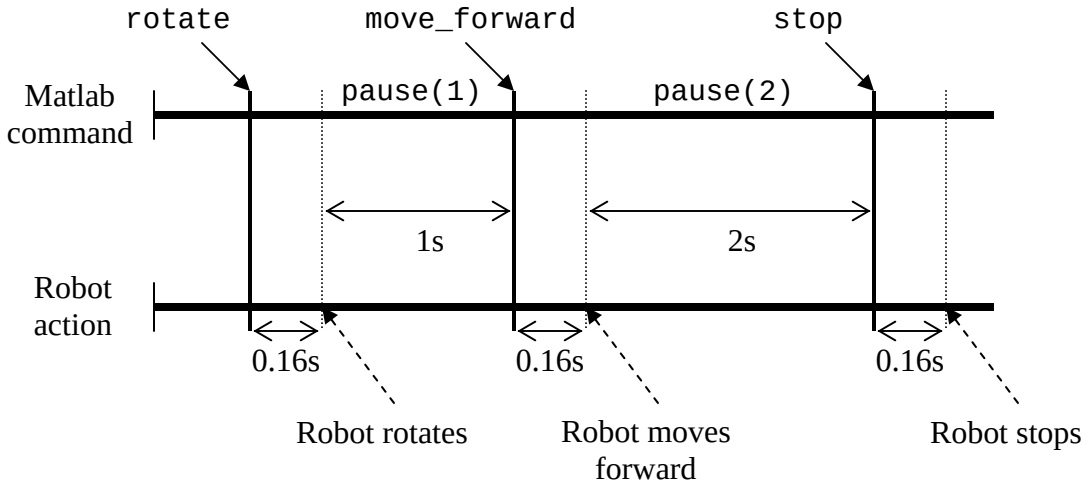


Figure 3.17. Representation of time delay incurred by wireless communication between Matlab (PC) and PBasic (robot).

To get around this issue, the duration of each rotation and forward move is scaled back by 0.16s. Although the error is not completely eliminated, its magnitude is now substantially reduced to several hundreths of a second.

Alternatively, the time duration of robot movement can be directly controlled by adjusting the elements in the PBasic `Pulsout` command. Section 3.4.3 describes how time duration of servos can be determined by specifying the number of iterations N of the `FOR` loop. However, PBasic lacks the capability for program interruption, implying that a `Pulsout` command has to be completely performed before the BASIC Stamp microcontroller can invoke the next input values from Matlab. Since Matlab timeline cannot be precisely synchronized with the BASIC Stamp, it becomes a huge programming ordeal, if possible at all, to ensure that the robot receives the input whenever one is sent from Matlab.

For experiment purposes, using Matlab `pause` command while scaling back the pause period to account for RF communication time delay should work adequately. The other problem is the fact that the PC is not running an RTOS (Real-Time Operating Systems). This makes the execution of any instruction non-deterministic, i.e., depending on background tasks, the execution time varies. In our setup, the PC is running with minimum background tasks to avoid overloading the processor.

3.5. EXPERIMENTAL RESULTS

Experiment in a real environment setting is performed to verify the assertions regarding the performance of the controllers. From the simulations, we expect the controller with the ANN and linear estimators the perform better than the reactive controller, as the formers are able to predict the object's position in the next time-step (or many time-steps ahead), while the latter is only capable of "reacting" to the real-time position of the object and conduct its pursuit in that direction. Since the ANN is a nonlinear estimator, capable of learning the behavior or movement pattern of the object after subjected to training, the ANN controller is expected to determine the path better hence outperforming the linear controller. The experiments were carried out with the following variations:

1. Type of controller: {reactive, linear, or ANN}
2. Camera device: {overhead or on-board)
3. Object path pattern: {circular or rectangular}

In these experiments, we are only analyzing object paths of circular and rectangular shapes. This is partly due to the tight confine of the lab which limits floor space for experimental use. Object paths such as circles and rectangles will maximize the utilization of floor area, as opposed to using infinity-shaped path, where turns are sharper, thus not allowing much room for movement for the predator. The robot used as the moving object is also a Boe-Bot, but without control access or communication with the

PC. The microcontroller in the robot has been programmed to govern the paths of the object. The pursued paths are as shown in Figures 3.18-20.

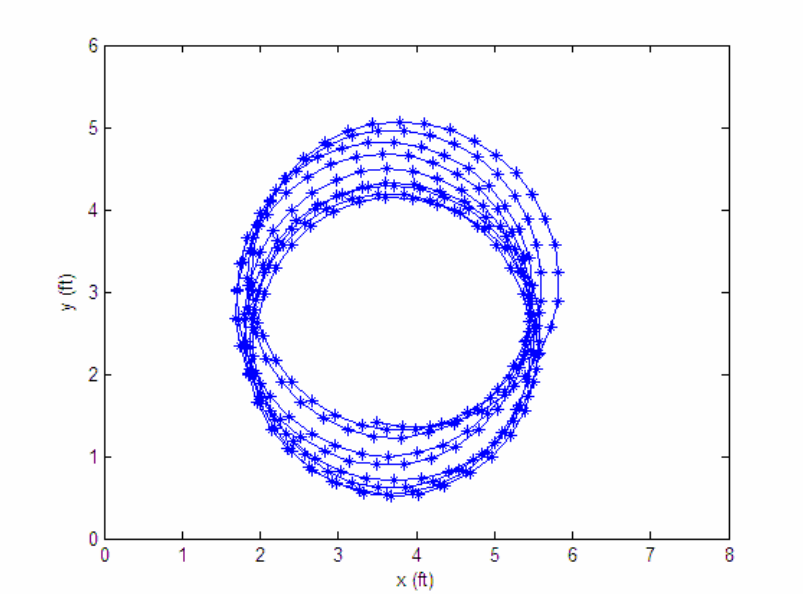


Figure 3.18. Circular prey path as captured by overhead camera. The effect of some noise from the playground terrain or servos is clearly seen.

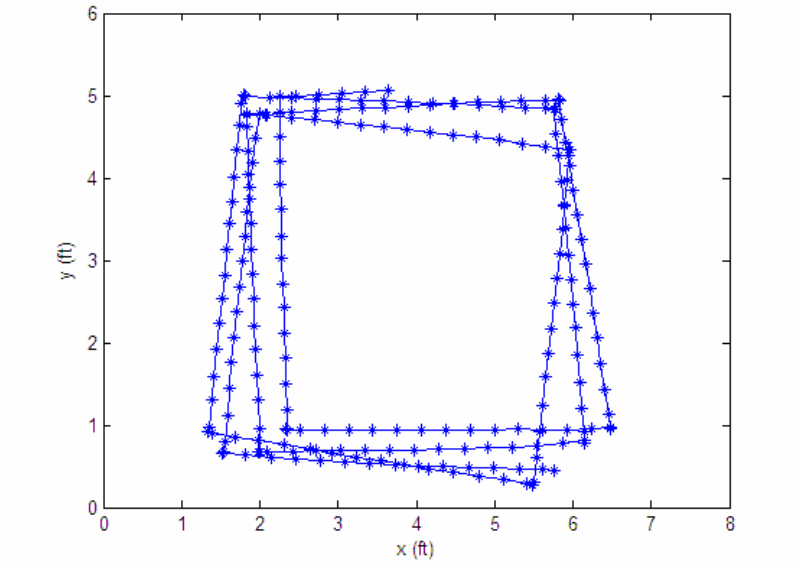


Figure 3.19. Rectangular prey path as captured by overhead camera. Notice the timing errors and noise in the sharp corners.

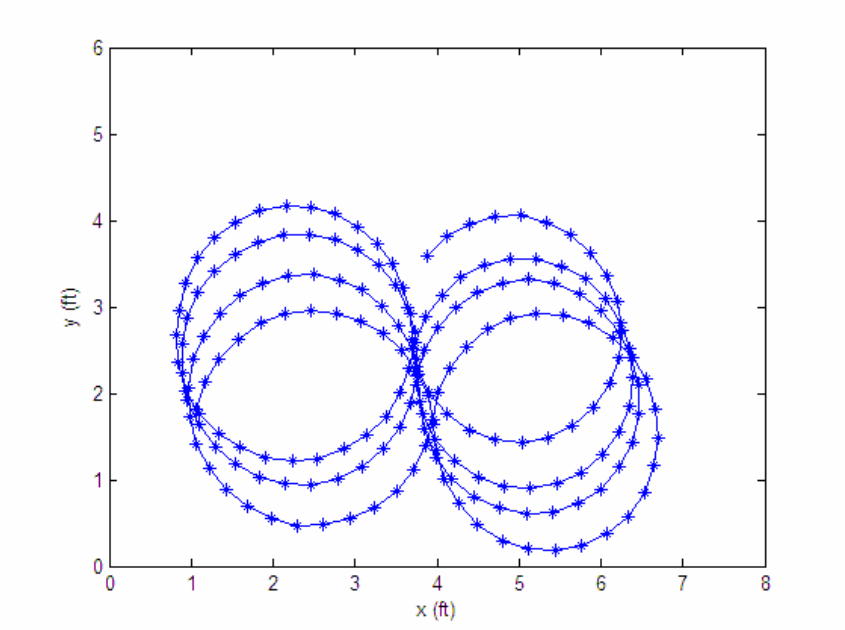


Figure 3.20. Infinity-shaped prey path as captured by the overhead camera. Notice the shift of paths due to noise from the environment (terrain or Boe-Bot's servos)

These paths are obtained by letting the prey run with movement routine downloaded onto its BASIC Stamp. As it moves around following the pre-programmed paths, the overhead camera captures the images. LabVIEW and the IMAQ Vision tools acquire and process these images to determine the position of the object continuously. It can be inferred from the plots that the movement of the object is not noise-free. Circle paths tend to shift over time, straight lines are not perfectly straight, and rotation angles when making turns -- in rectangular pattern -- are not precise either. This inconsistent behavior of servos and variation in slip/friction and timing is observed throughout the experiments in both the prey and predator motion.

3.5.1. Experiments with Overhead Camera as Image Capture Device

3.5.1.1. Tracking Circular Pattern

The color identification tag of the markers placed on top of both the prey and predator is approximately 8” in diameter. As such, the closest distance that they can get to each other is roughly 8” before colliding and stopping their movement. This constrains the error tolerance to about 8”. Such large error tolerance value becomes a factor that makes interception a common occurrence, regardless of which control algorithm is in use. As a result, for performance analysis, the fitness criterion used is interception time. For an “apples-to-apples” comparison of the experiment results, they must have the same (or similar) initial conditions before these results can be used for performance analysis. If the initial conditions are varied, one experiment may falsely suggest a better performance than others, i.e. shorter interception time, although the difference may have been caused by small initial errors. The chosen condition is when the initial error is +/- 2ft.

Fig. 3.21-22 present the results – mapping of pursued path and error plot -- of the predator as it attempts to intercept the prey which is moving in a circular pattern. The predator exhibits the reactive controller in this first result. Green circles indicate position of the prey, while red asterisks for the predator. The prey moves in clockwise direction and so does the predator as it attempts to chase the prey.

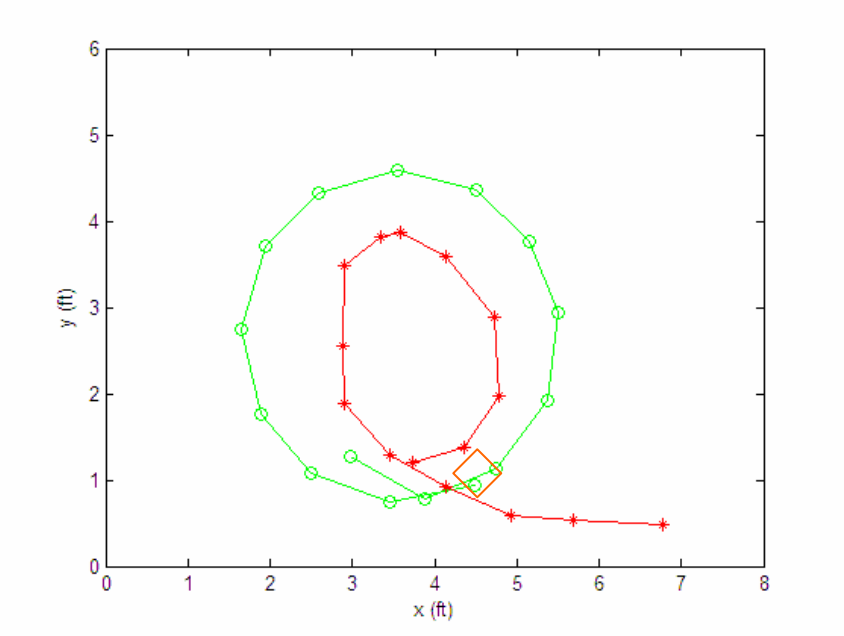


Figure 3.21. Plot of the paths of the reactive controller when using overhead camera with the predator robot (red) pursuing the prey (green) which moving in circular pattern. Shown with a diamond is the initial condition of the prey.

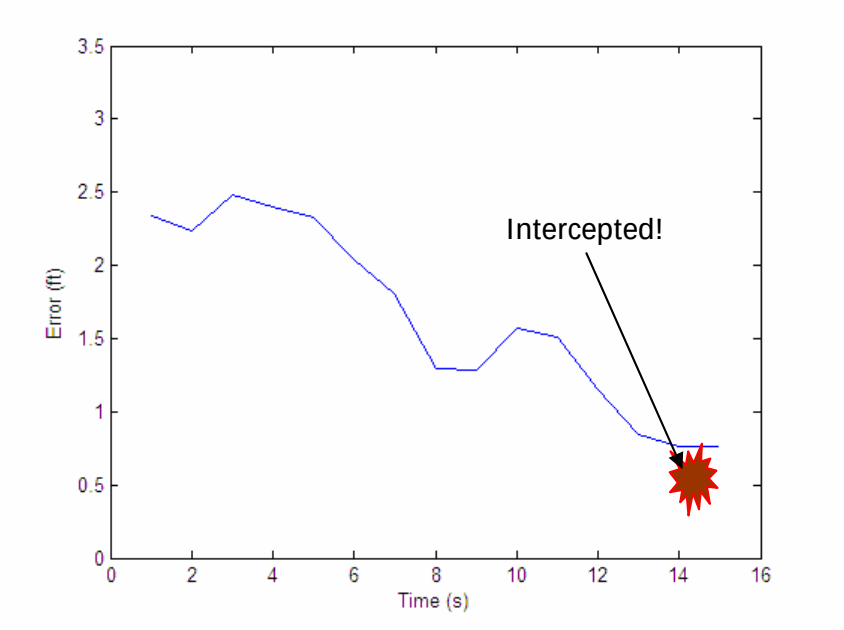


Figure 3.22. Error plot of reactive controller when using overhead camera.

Prey interception is achieved when the error magnitude is approximately 0.7 ft (8 inches). From Fig. 3.22, the interception time is 14[s]. Another point for discussion is the initial error in this experiment which is larger than the prescribed 2ft. Considering the difficulties involved in ensuring initial distance error between the prey and predator, this experiment's result is used for analysis although its initial error magnitude is 2.4ft. From the error plot at t = 10-11[s], there is a noticeable jump in error value. This is the point in the path plot where the predator "has just realized" that the prey is making a circular turn. The surprise effect renders the predator to make a rather sharp turn, and as predator re-orients to correct its direction, the prey keeps moving away and thus error is increased.

With linear estimator controller, the interception time is expected to be reduced as linear controller is predicted to be a better controller (see Figures 3.23-24).

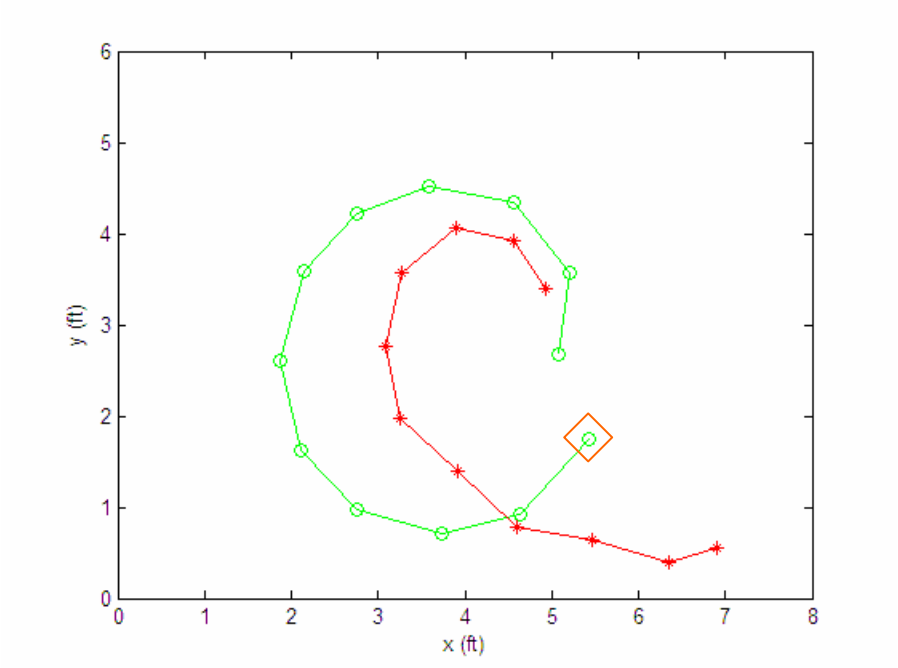


Figure 3.23. Path plot of linear controller when using overhead camera with the robot (red) pursuing the object (green) which moving in circular pattern.

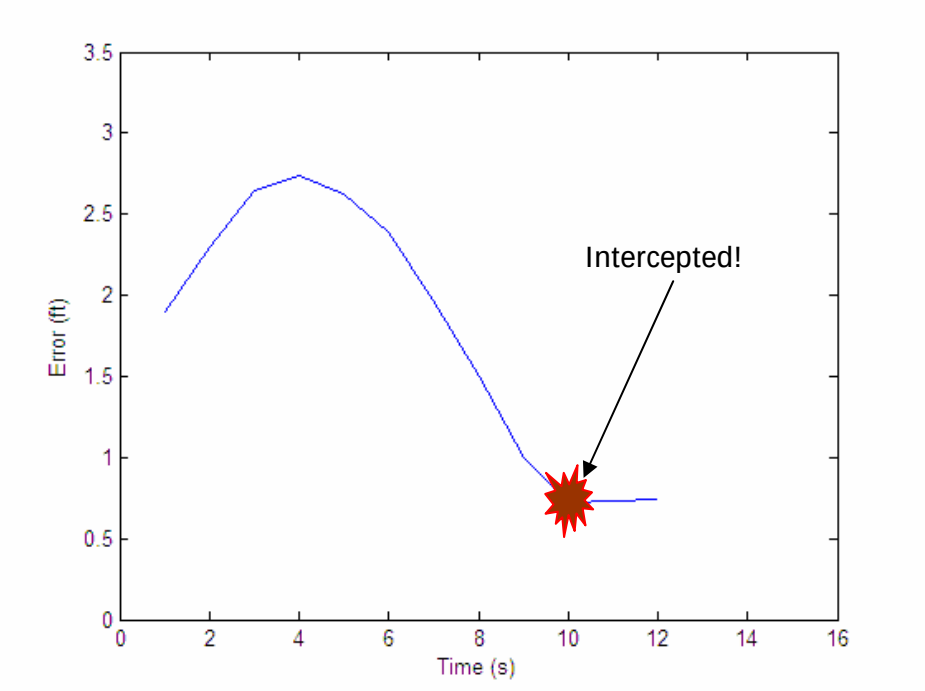


Figure 3.24. Error plot of linear controller when using overhead camera.

Predictably, the linear controller improves the robot performance as interception time is reduced to 10[s]. This is due to controller's ability to predict where the object would be in the next time-step. This characteristic of linear controller is exhibited in the path plot where the robot evidently approaches the object not by moving directly towards it, but towards the coordinate one time-step ahead of the object, before finally making the interception. The error plot shows an initial hump of the error; this is because the linear estimator algorithm works by estimating the speed of the prey, which is done by comparing the prey current and previous position. As such, the predator is actually not moving in the first time-step. This can be improved by using reactive controller for the first time-step, instead of staying put. Although not visible in the path plot, there are two

overlapping points at the initial position of predator. As a result, the error increases sharply in the first time-step because the predator is stationary while the prey is moving away from it.

The last controller under study is the ANN controller which is hypothesized (from the simulation results) to be a better estimator and hence a better controller than both reactive and linear controller. The implementation of ANN controller to intercept circular path is illustrated in Figures 3.25 and 3.26.

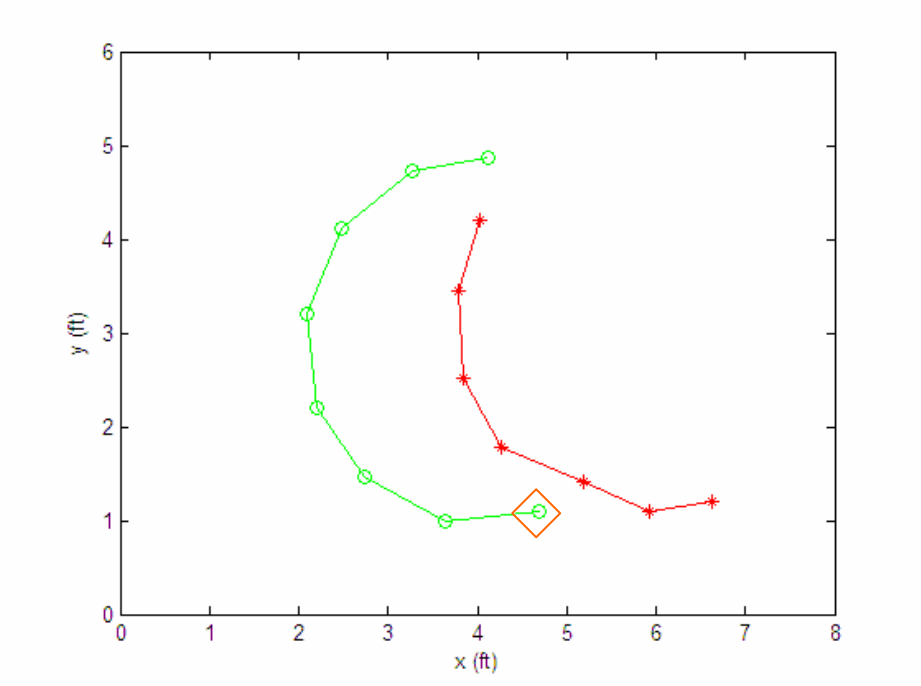


Figure 3.25. Path plot of ANN controller when using overhead camera with the robot (red) pursuing the object (green) which moving in circular pattern. The initial position of the prey is marked with the red diamond.

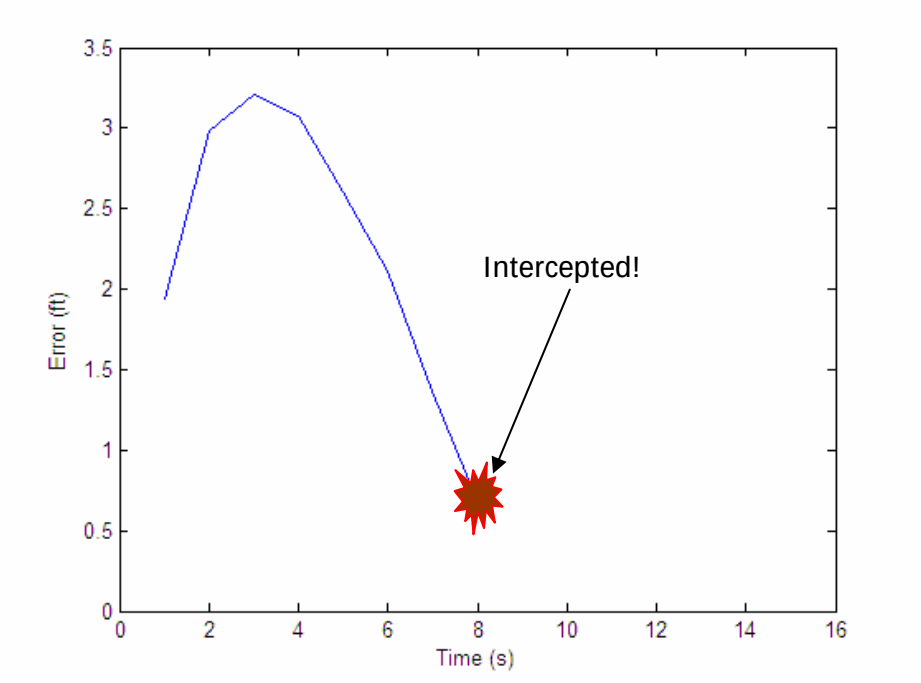


Figure 3.26. Error plot of ANN controller when using overhead camera for object moving in circular pattern. Interception is achieved in 8[s].

Prior to experimental simulation, the ANN controller first has to be trained with a set of data set of prey coordinates when moving in circular pattern as shown in Fig. 3.17. After successful training, mapping relationship between the input and output data is generated and used for simulation. The ANN controller is more superior to the reactive and linear controller as deduced from the significant reduction in interception time. Path determination and estimation of the prey in the future time-step are calculated with enough precision, and interception takes place in just 8[s]. Similar to the linear controller's error plot, the initial error hump is also observed when ANN controller is used. This sharp increase in error is caused by the wait time that the robot has to undergo to gather data for path estimation. Once the robot moves, its direction is governed by the

nonlinear neural network algorithm that precisely estimates the future coordinates of the prey, more precise than the linear estimator does. As a result, prey interception is accomplished in a shorter time period.

3.5.1.2. Tracking Rectangular Pattern

A variation to the initial experiment is introduced by using a different prey's path. Fig. 2.27-29 show the path and error plots results when the object moves in a rectangular pattern, and the robot's pursuit algorithm is varied from reactive to linear and to ANN controllers. As in the case for circular path experiments, the pursuit runs with rectangular path is conducted for each controller. The initial condition used to select the most suitable results for comparison is initial error magnitude of +/-3.5ft.

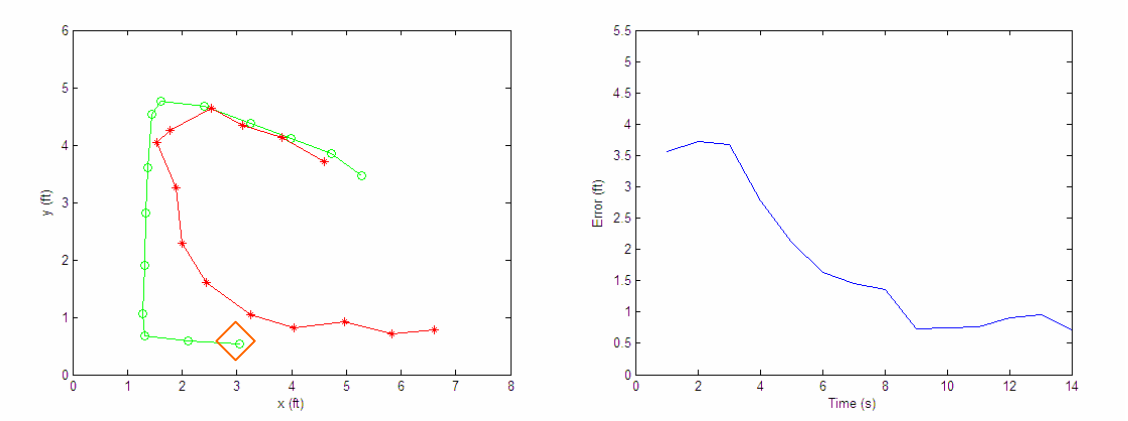


Figure 3.27. Path and error plots of reactive controller following a rectangular path

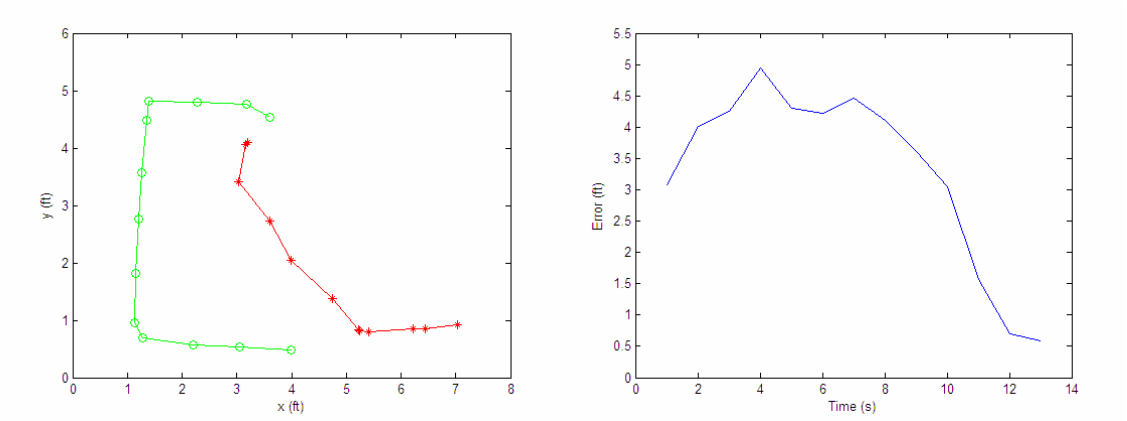


Figure 3.28. Path and error plots of linear controller following a rectangular path

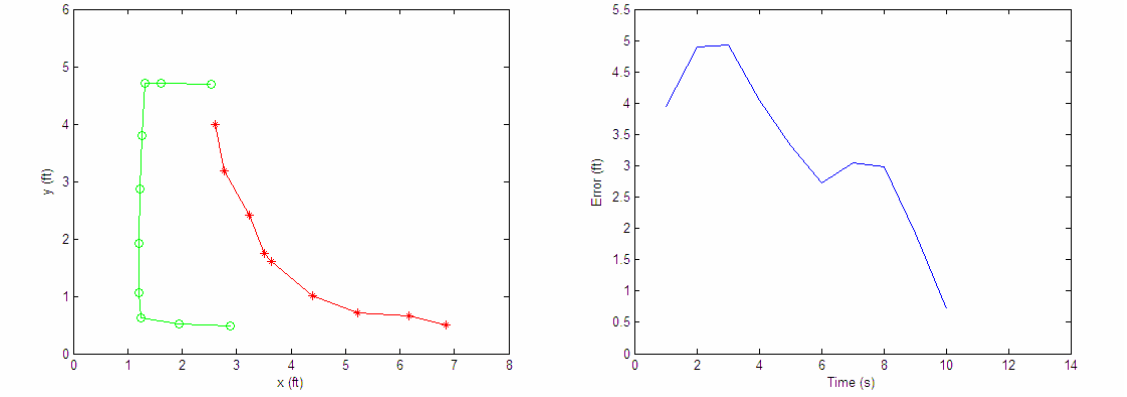


Figure 3.29. Path and error plots of ANN controller following a rectangular path

From the path plot of the reactive controller in Fig. 2.27, the inconsistency of prey's servos can be observed at the upper portion of the rectangle, where it tapers downwards instead of moving horizontally to the right. Nonetheless, the reactive controller is able to cope with this problem. The predator closely follows the prey's tail, about 1[s] behind, before finally intercepting the prey at t = 14[s].

The results for the linear estimator are given in Fig. 2.28. This controller helps the predator intercept the prey in 13[s], a little quicker than the reactive controller. Again, this is due to the nature of linear controller that estimates the next position of the prey by linearly extrapolating from the prey's current and previous coordinates. To accommodate this, however, the controller will first have to wait for one time-step and the robot stay motionless so as obtain knowledge of the object's speed. This results in an increased error in the first second as the object is moving away from the stationary robot. This error is further increased in the subsequent time-steps due to inconsistent robot servos, where the forward move is unexpectedly halted prior to completion. This occurrence can be observed in the path plot. Nonetheless, despite the erroneous servos behavior, linear controller still successfully intercepts the prey in a shorter time than reactive controller does.

Better performance is achieved with ANN controller -- shown in Fig. 2.29 -- as the interception time is reduced to 10[s]. The input data set used for training by the neural network is the coordinate matrix of prey in Fig 2.19. The initial error build-up is also observed in the error plot, a characteristic for estimator controller. The initial error in this experiment is larger than that when using reactive and linear controller: approximately 4ft, as opposed to 3.5ft and 3ft for reactive and linear controller, respectively. Robot servos inconsistencies also take place during the experiments, specifically at t = 5[s], as seen from the path plot. Nonetheless, despite the initial error build-up, larger initial error magnitude, and servos flaws, the ANN controller still prevails and intercepts the prey more quickly than the other two controllers.

3.5.2. Experiments with On-board Camera as Image Capture Device

As far as the control algorithms are concerned, the experiments are identical in nature when overhead or on-board camera is used as the image acquisition device. The only difference is in the hardware setup of the camera itself. Upon successfully selecting a compatible wireless camera, the camera is mounted on the robot by means of metal frames. The key installation aspect to note is that the camera lens has to be vertically aligned with the center of rotation of the robot. This is to prevent robot's vertical vision line from shifting as the robot rotates. In Section 3.2.5.1., the determination method of object coordinates for continuously-changing coordinate frame has been established. In the mathematical derivation, it is assumed that robot rotation only introduces angular displacement, not translational.

For successful image processing, the object has to be placed in front of a background of contrasting color. This is accomplished by draping the surrounding background wall with matte, black cloth. Due to camera's sensitivity to light reflection and over-exposure, ceiling lights have to be turned off. Fig. 3.30 shows the playground's setting with the predator and prey in the field.

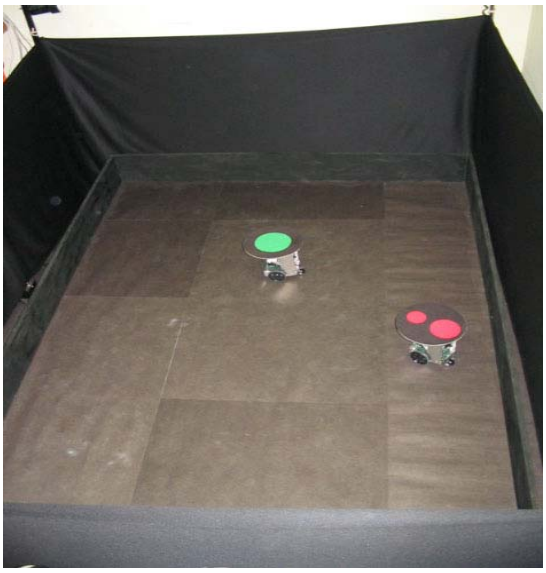
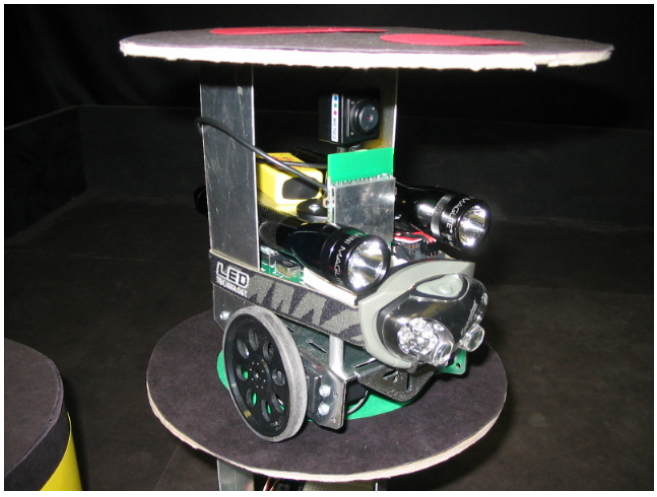
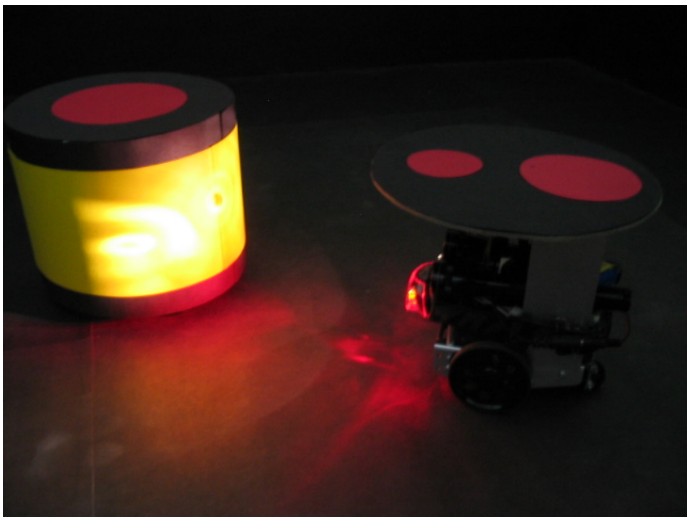


Figure 3.30. Playground's setting for experiment with on-board camera, shown with the predator (red markers) and prey (green marker) in the field. The black cloth draped against the wall is to ensure a dark background, as seen by the predator through the on-board camera.

Spotting flashlights and night-vision headlight are mounted on the predator as the light source (Fig. 3.31-a). The prey itself is enclosed with colored paper, which is cut to consistent height (Fig. 3.31-b).



(a)



(b)

Figure 3.31. Close-up images of the predator and prey in the "on-board camera" experiment setting. (a) The predator is shown fully-equipped with the RF antenna, flashlights, night-vision headlight, and the wireless on-board camera. (b) The prey is hidden under the cover during the experiment. The predator identifies the prey as a yellow object, and its relative position is determined using the algorithm detailed in Section 3.2.5.1.

Upon image filtering, the relative (x,y) position of the object is related to the height and position of the colored paper (Section 3.2.5.1), as seen by the predator through the camera vision window. Figure 3.32 shows the various hardware components used to facilitate the use of on-board camera as image capture device.

Figure 3.32. Hardware needed for image acquisition using on-board camera. Shown in the picture are (from left to right) the flashlights and headlight, DVD Maker USB2.0 Capture Box, Q-See 4-Channel Receiver, and Q-See 2.4 GHz Wireless Camera (bottom).

The implementation of the on-board camera as image input source proves to be a challenging task. The field-of-view of the camera is +/- 47 degrees, which becomes a major source of limitation to the functionality of the camera. To work around this limitation, the experiment is conducted more slowly, i.e. with prey's and predator's speeds reduced by half. As detailed above, and similar to the experimental setting where the overhead camera was used as the image capture device, various hardware difficulties entails the experiment with the on-board camera. As such, the results presented in this section are the most comprehensible ones amongst all the conducted experiments. The

predator will be controlled to track the prey which moves in circular pattern, and the results are presented in Figures 3.33-35.

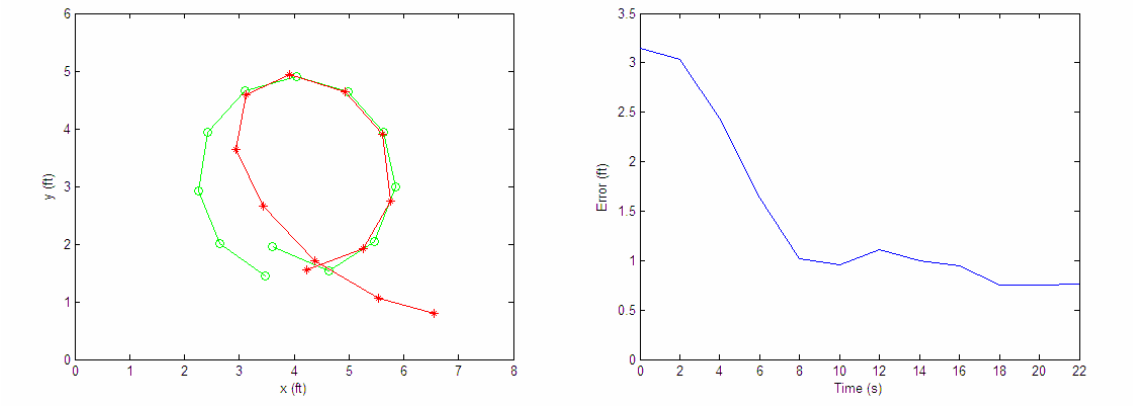


Figure 3.33. Path and error plots of reactive controller following a circular path.

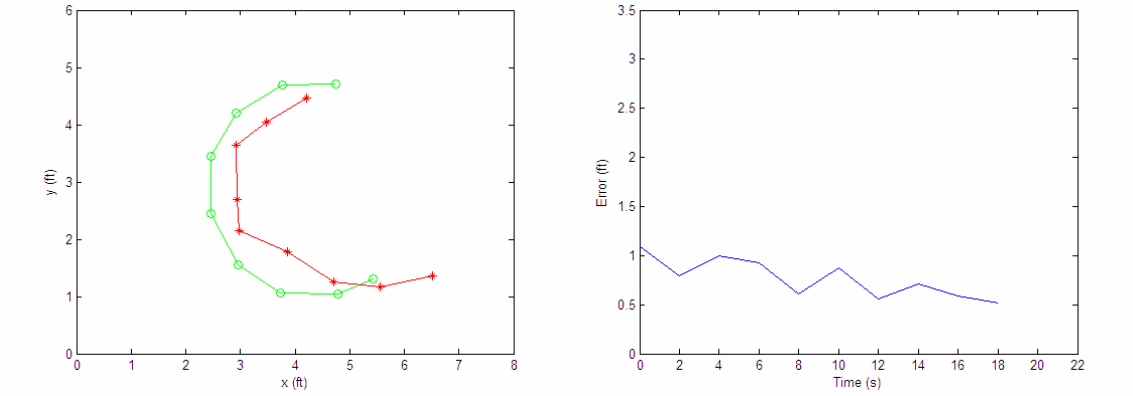


Figure 3.34. Path and error plots of linear controller following a circular path.

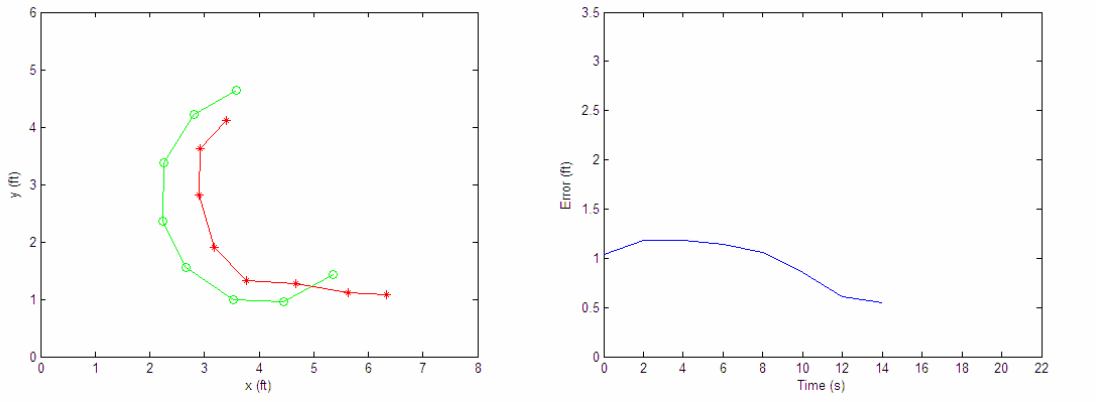


Figure 3.35. Path and error plots of ANN controller following a circular path.

Throughout these experiments, the time-step size is 2[s], instead of 1[s] as in the overhead camera experiments. The chosen initial error value for an experiment result to be accepted for analysis is 1ft. For the reactive controller, however, the result used has an initial error of 3.1ft, significantly greater than the accepted value. Compared to other results, however, this is by far the most comprehensible set of data for the reactive controller. From Fig. 3.33, the interception time is determined to be 22[s], although this information is of questionable use, as far as performance analysis is concerned. The path plot in Fig. 3.33 depicts a typical behavior of reactive controller, where it trails the object one time-step behind; the robot aims to be where the prey is a time-step earlier.

The results for the linear and ANN controller have many similarities as far as the path taken by the predator in its pursuit to intercept the prey. The initial errors in both experiments are around 1ft, rendering them acceptable for an “apples-to-apples” (equal basis) analysis. The interception time for the linear controller is 18[s] (Fig. 3.34) while that for the ANN controller is 14[s] (Fig. 3.35). This improved performance of the ANN

controller is due to its ability to learn to estimate the nonlinear behavior of the prey path during the training phase. Although the path plots in Figures 3.34 and 3.35 are generally similar, the estimation in the former result is less accurate in predicting the prey's position, as inferred from the path plots and error fluctuations in the error plot of the linear controller.

Chapter 4 – Summary and Conclusions

4.1. SUMMARY

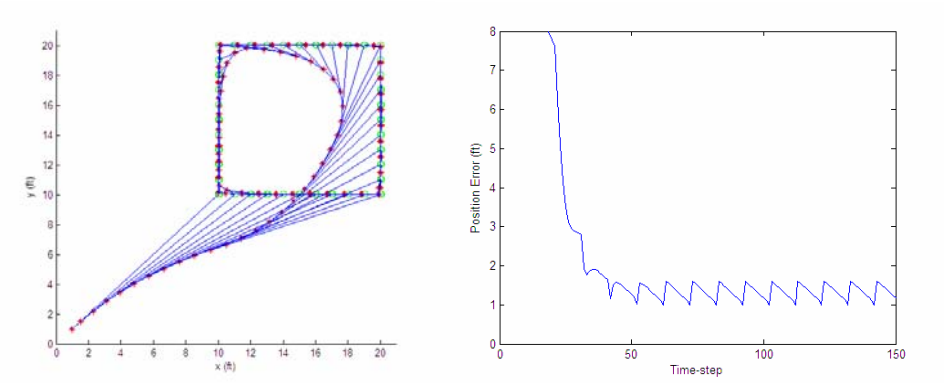
This study investigated the performance of several control techniques for a mobile robot (predator) to autonomously track and pursue a moving object (prey). The predator-robot's path was calculated using three different techniques: reactive controller, a predictive controller with linear estimation, and a predictive controller using an artificial neural network (ANN) estimator. The derivations and architecture of each control algorithm are detailed in Chapter 2. A computer model was developed to predict the path taken by the predator as it tracks the prey moving using different behaviors (paths).

In the setting of predator-prey pursuit environment, the pursuer requires a robust controller to guide its path by determining its direction in real-time, until interception is accomplished. Motivated by various applications of such technology, this study explores different control algorithms and determines the fittest controller that is most suitable for use in pursuit environment. Fitness of each method was based on the interception time.

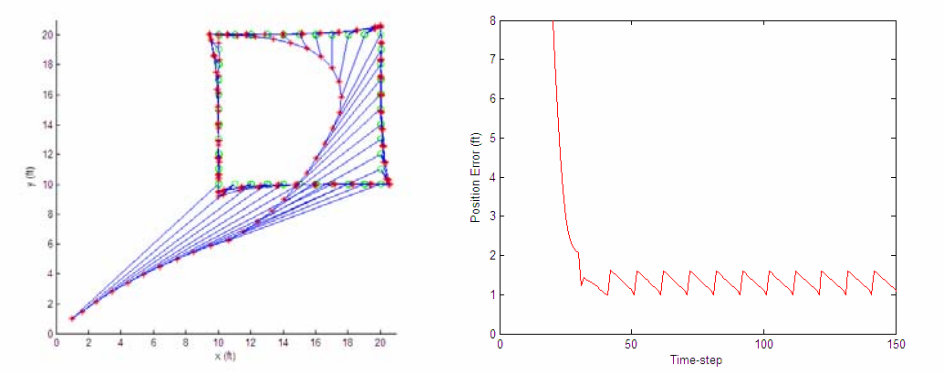
The reactive controller "reacts" to the current position of the prey and guides the robot to move in that direction (discussed in Section 2.1). The predictive controller with linear estimator (discussed in Section 2.2) estimates the velocity of the prey and its position in the next time-step, then uses this as the target position. The predictive controller based on the ANN estimator generates a nonlinear model of object's behavior and accurately estimates the future position(s) of the prey, based on the object positions

several time-steps earlier. The specific type of neural network used in this study is a multi-layer perceptron with supervised learning algorithm, discussed in Section 2.3.

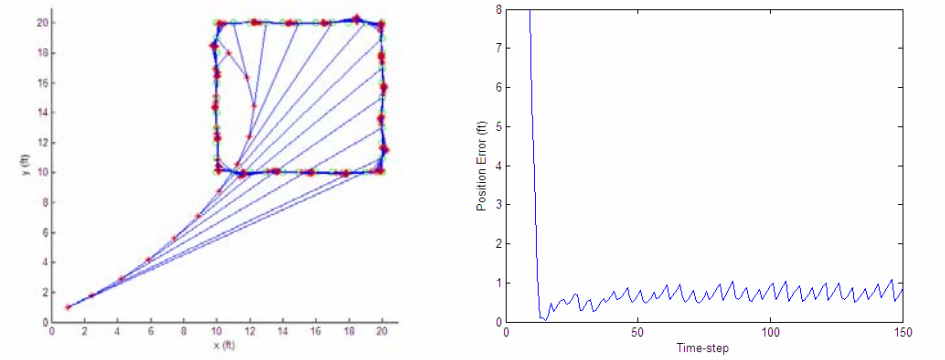
Computer modeling and simulations were performed in Chapter 2 to predict the behavior and performance of each controller. A number of simulation parameters, such as prey path and time-step, were varied to observe the performance of the controllers in different environments. The prey is simulated following a circular, rectangular, infinity-shaped, and random path. Two time-step values were chosen, in one, the predator is slower than the prey, and in the other, their speeds are approximately equal. In all of the computer simulations, no controller is able to intercept the object in a real sense that distance error is reduced to zero due to the computational delay to react and the lack of sufficient speed to catch the prey. At steady-state, the predator attained an error magnitude which is either constant or regularly fluctuating. For performance analysis purposes, the fitness criterion chosen was the steady-state error magnitude. With this criterion, the ANN controller proves to be the best controller, regardless of the simulation setting. The following figures are reproduction of the simulation results from Chapter 2 when the controllers are used to guide the robot to pursue an object following a rectangular path.



(a) (b)



(c) (d)



(e) (f)

Figure 4.1. Matlab simulation results for reactive, linear, and ANN controllers with time-step 2.3[s].

The results above are for the simulations where time-step is 2.3[s], which corresponds to the robot having approximately equal speed as the object. The ANN (-based) controller is shown to perform better than the other two as the steady-state error magnitude is decreased rapidly in 10 seconds, and it attains a steady-state error behavior at the 12^{th} time-step. This improvement in error reduction was achieved by the ANN controller because an ANN is a nonlinear estimator capable of learning the behavior of the prey's movement. The linear estimator, although superior to the reactive controller, falls short when compared to the ANN controller in accurately estimating the future position of the prey, especially when the prey makes sharp turns at corners. This occurrence is notable in Fig. 4.1-(c) where the predator curves out slightly every time it turns a corner of the rectangle.

In Chapter 3, the computer models are validated through implementation of the control algorithms in a real-world environment. To facilitate experimental simulations, various tasks need to be accomplished. These included developing an image acquisition and processing system, where the required hardware and software need to be fully integrated and able to function in real-time to provide information from the field to the robot. The hardware setup include:

- The predator and prey robots: mobile robots manufactured by Parallax Inc., augmented with radio transmitters. The robots communicate with a PC (Dell Inspiron I5160, Intel Pentium 4 CPU 3.06 GHz, 1.00 GB RAM) that controls the prey and predator robots behavior.

- The LPS (Local Positioning System). An overhead camera vision system that provides feedback to the robots about their local positions and to the performance evaluation system.
- An on-board camera in the predator robot. Using on-board camera, coordinate frame correction algorithm needs to be performed to calibrate each calculation to the true coordinate reference of the field.

The image processing software used in the experiments are LabVIEW and IMAQ Vision, used with NI-IMAQ for USB Cameras add-on software for the camera to be recognized by LabVIEW. The image processing algorithm allows for a robust object identification and position determination. The coordinate data from LabVIEW are passed as system inputs to Matlab via ActiveX server application. The necessary rotation and position corrections are estimated, and action commands are transmitted to the robots via the RF communication. The robot itself needs to be built, tested, and programmed to receive and execute the wireless signal from PC. The "brain" of the robot is an integrated microcontroller that is programmed in PBasic computing language. Any movements made by the robot (and the object) will be captured by the camera and used as input data to the system again. This algorithm loops continuously while the supporting hardware (camera, Boe-Bots, RF transmitter and receiver, PC) and software (Matlab, LabVIEW, IMAQ Vision, PBasic) operate simultaneously, until interception is accomplished.

During the experimental simulations, several parameters were varied, including the controller's type, prey path, and camera position. For analysis on equal basis, the initial condition of the experiments needed to be standardized; in the experiments, the

chosen initial condition is the distance error between prey and predator. Fig. 4.2 shows the results from the experiment, as the robot pursues the rectangular path of the object.

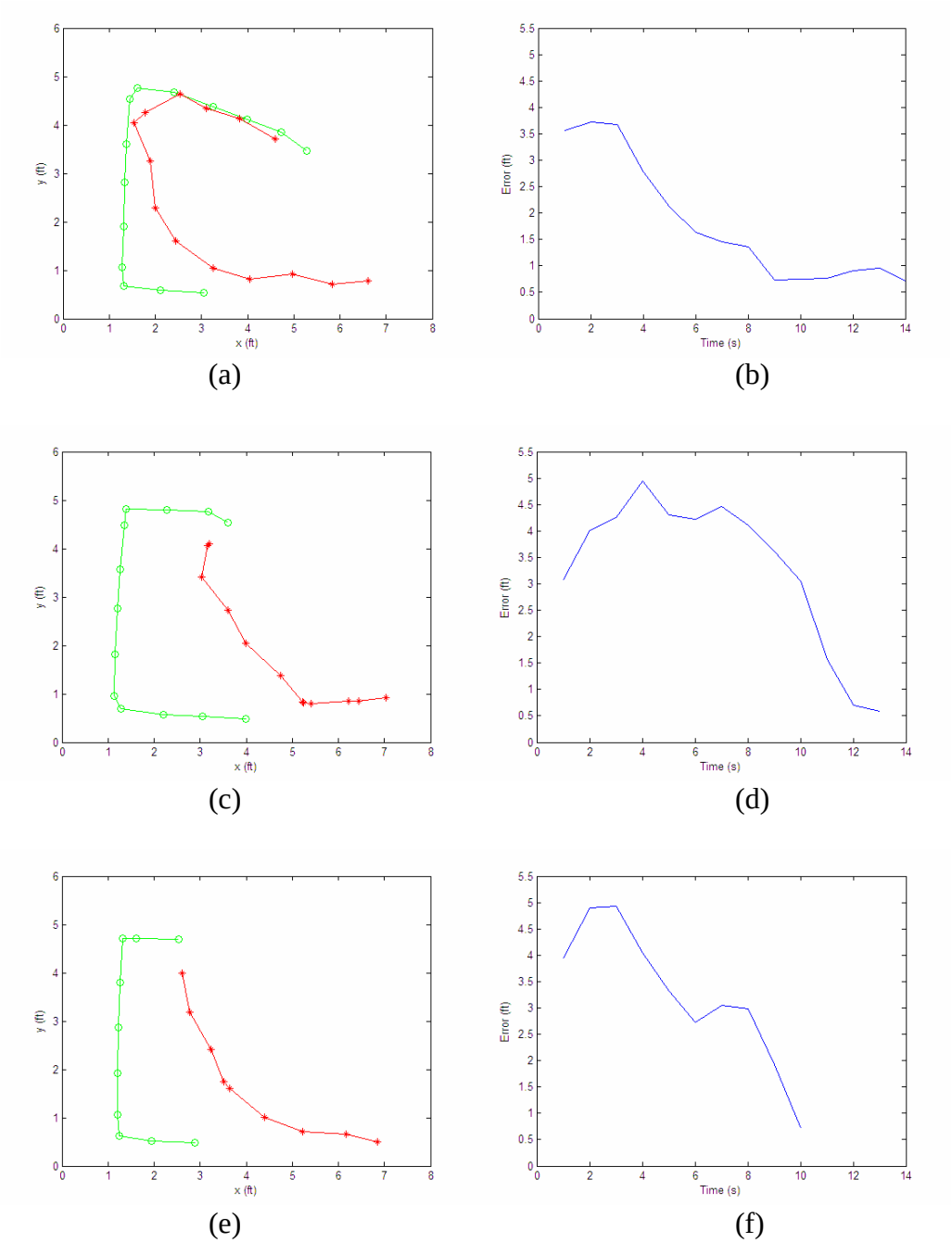


Figure 4.2. Real-world experimental results for reactive, linear, and ANN controllers using overhead camera.

Overall, through these experiments, we can conclude that the ANN controller is the best controller as it achieves the simulation objective, i.e. prey path interception, within the shortest amount of time. Detailed analysis of the results is presented in Section 3.5. This conclusion, however, is only true given two conditions:

1. Data sets are available for training the neural network
2. The prey behavior stays within the set of patterns used for training

These conditions are generally fulfilled in this experiment as the network is always trained prior to being simulated with a new path, and all the paths have certain patterns inherent to themselves that can be recognized by the network. The random path that is simulated in the computer is not experimented in the real-world scenario. However, when random path is included, the reactive controller will likely be the favored controller. Although the reactive controller may not bring about a better performance, considering the simplicity of its algorithm and its low computational load, it will be able to perform the computation much faster than the linear and ANN estimator controllers, thus more suitable in a fast-paced simulation of random pursuit of object.

4.2. FUTURE DIRECTIONS

While predictive controller with ANN estimator proves to be the best controller for use in the pursuit environment, as shown by the results in this study, its applicability in real-world scenario -- for example, anti-ballistic missile technology -- is questionable as training phase may not be available prior to deployment. To retain the superiority of

neural network estimator while enhancing its applicability, the class of ANN algorithm -- supervised learning -- may be replaced by one that does not require training, for example the reinforcement learning model discussed in Section 2.3.1. In its operation when training is no longer required, there is a possibility

After determination of the pursuit direction, the robot follows two movement steps: reorientation and forward move. This error correction routine is not the most time-efficient, as the steps are performed sequentially. As a result, when the robot is rotating, its error steadily increases as the object moves away. An improvement to the system can be achieved by combining the two movements together. Instead of sending action command to the servos to rotate with an equal, opposite angular velocity to perform rotation, followed by equal angular velocity to move forward, the servos can be controlled to rotate asymmetrically by adjusting the timing of the voltage pulses sent to each servo. The action will use maximum speed in the wheel opposite to the desired turn and the other wheel adjusted to turn the desired angle and then go full speed. This will result in a curved, one-step maneuver of the robot, much like the path of a vehicle as it moves along a curved road. This scheme will entail a design of a sub-controller to govern the pursuit maneuver of the robot. A sub-optimal controller solution can be incorporated in this system, with time-to-intercept being the parameter to be minimized in the cost function.

For future works, hardware upgrade is an area that has several rooms for improvement. The Boe-Bot servos, in particular, are very inconsistent and unreliable at times. Replacing these with ones with encoders is likely to bring about improvement.

Multiple cameras can be installed on the robot to give a wider field-of-view to the robot. The wireless communication between the PC and Boe-Bot can also be replaced with BlueTooth communication instead of RF. BlueTooth technology has the potential of reducing or eliminating time delay between the PD and BoeBot.

Appendix A

Matlab Source Code for ANN Multi-Layer Perceptron Training

MLP.m

```
clear all
clc


% input & output initialization
M = circ;
inbatch = 9;        % number of elements per input batch
outbatch = 4;       % number of elements per output batch
out_est_count = 2;  % number of useful estimated output values
% index different between input & output
inout_diff = out_est_count + inbatch - outbatch;
% index increment between timestep
input_incr = 2;

% input & output assignment using M
for i = 1:30
    in = i + (i-1)*(input_incr-1);
    out = in + inout_diff;
    input(i,:) = [M(in:(in+inbatch-1),1)' M(in:(in+inbatch-1),2)'];
    output(i,:)   =   [M(out:(out+outbatch-1),1)'   M(out:(out+outbatch-
1),2)'];
end

% reduce input & output values

% NN parameter initialization
I = inbatch*2;
H = 20;
O = outbatch*2;
eta = 0.001;
MaxEpochs = 50000;



% NN training
[e,epoch,W1,W2,o2]=MLP_sim (M,I,H,O,eta,MaxEpochs,input,output);

% NN simulation
input_sim = input + rand(size(input))*.05;  % noise added
P = size(input_sim,1);
a = [input_sim,ones(P,1)];
h1 = a*W1;
h2 = logsig(h1);
b = [h2,ones(P,1)];
```

```
o1 = b*W2;
o2 = o1


% i2x is input values for NN
for i = 1:30
    i2x(((i-1)*inbatch+1):i*inbatch,1) = input_sim(i,1:inbatch)';
    i2x(((i-1)*inbatch+1):i*inbatch,2)=
input_sim(i,inbatch+1:inbatch*2)';
end

% o2x is output values from NN, i.e. estimated prey position
for i = 1:30
    o2x(((i-1)*outbatch+1):i*outbatch,1) = o2(i,1:outbatch)';
    o2x(((i-1)*outbatch+1):i*outbatch,2) = o2(i,outbatch+1:outbatch*2)';
end
```

MLP_sim.m

```
function [e,epoch,W1,W2,o2] =
MLP_sim(M,I,H,O,eta,MaxEpochs,input,output)
% M is the matrix with inputs and outputs together
% I is the number of inputs
% H is the number of neurons
% O is number of outputs
% MaxEpochs = 5000 maximum number of epochs


% Initialization
W1 = rand([I+1 H]);   % I+1 for bias' weight
W2 = rand([H+1 O]);   % H+1 for bias' weight
P = size(input,1);
epsilon = 0.001;
epoch = 1;
e = ones(P,O);


while (sum(e.^2) > epsilon) & (epoch < MaxEpochs);

    dW1 = zeros(size(W1));
    dW2 = zeros(size(W2));

    a = [input,ones(P,1)];
      h1 = a*W1;
      h2 = logsig(h1);

      b = [h2,ones(P,1)];
      o1 = b*W2;
      o2 = o1;

      e = o2 - output;

    do2_do1 = ones(P,size(output,2));         % f'ok(ukp)
```

```
    dh2_dh1 = dlogsig(h1,h2);     % f'hj(ujp)

    dop = e.*do2_do1;             % ekp*f'ok(ukp)
    dWjp = (dop'*b)';             % del_wkj

    Wjp = dop*W2(1:H,:)';           % summation K
    dVji = a'*(dh2_dh1.*Wjp);       % del_vji

    dW1 = dW1 - dVji;
    dW2 = dW2 - dWjp;

    W1 = W1 + eta*dW1;
    W2 = W2 + eta*dW2;

    epoch = epoch + 1;

end
```

Algorithms	INPUTS			OUTPUTS	BEP OUTPUTS
XOR	0	0		0	0.010
	0	1		1	0.985
	1	0		1	0.986
	1	1		0	0.021
N-Parity	0	0	0	0	0.002
	0	0	1	1	0.998
	0	1	0	0	0.021
	0	1	1	0	0.022
	1	0	0	1	1.003
	1	1	0	0	0.005
	1	1	1	1	0.995
sin(x)sin(y)	-1.5	-1.5		1.0	1.002
	-1.5	-1.0		0.0	0.022
	-1.5	-0.5		-1.0	-0.996
	-1.5	0.0		0.0	0.022
	-1.5	0.5		1.0	1.130
	-1.5	1.0		0.0	0.021
	-1.5	1.5		-1.0	-0.996
	-1.0	-1.5		0.0	0.021
	-1.0	-1.0		0.0	0.022
	-1.0	-0.5		0.0	0.005
	-1.0	0.0		0.0	0.021
	-1.0	0.5		0.0	0.022
	-1.0	1.0		0.0	0.005
	-1.0	1.5		0.0	0.022
	-0.5	-1.5		-1.0	-0.996
	-0.5	-1.0		0.0	0.022
	-0.5	-0.5		1.0	1.002
	-0.5	0.0		0.0	0.005
	-0.5	0.5		-1.0	-0.996
	-0.5	1.0		0.0	0.005
	-0.5	1.5		1.0	1.002
	0.0	-1.5		0.0	0.010
	0.0	-1.0		0.0	0.021
	0.0	-0.5		0.0	0.021
	0.0	0.0		0.0	0.022
	0.0	0.5		0.0	0.005
	0.0	1.0		0.0	0.010
	0.0	1.5		0.0	0.021

	0.5	-1.5	1.0	0.991
	0.5	-1.0	0.0	0.005
	0.5	-0.5	-1.0	-0.996
	0.5	0.0	0.0	0.005
	0.5	0.5	1.0	0.991
	0.5	1.0	0.0	0.005
	0.5	1.5	-1.0	-0.996
	1.0	-1.5	0.0	0.022
	1.0	-1.0	0.0	0.005
	1.0	-0.5	0.0	0.010
	1.0	0.0	0.0	0.021
	1.0	0.5	0.0	0.010
	1.0	1.0	0.0	0.021
	1.0	1.5	0.0	0.010
	1.5	-1.5	-1.0	-0.996
	1.5	-1.0	0.0	0.005
	1.5	-0.5	1.0	0.991
	1.5	0.0	0.0	0.005
	1.5	0.5	-1.0	-0.996
	1.5	1.0	0.0	0.003
	1.5	1.5	1.0	0.971

Appendix B

Configuration of SureLink & QuickLink Wireless RF Modules

The most important aspect of our experiment proved to be the communication between the Boe-Bot and the computer running our MATLAB applications. The final experiment required us to be able to communicate wirelessly, however we first needed to understand the basics of simple wired communication. Therefore, we first set about getting the Boe-Bot and the computer to communicate using DB-9 serial chords.

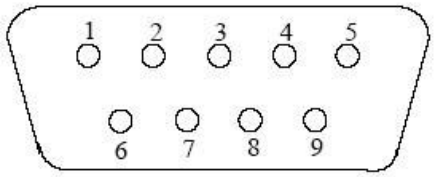


Figure 1. A DB-9 serial cable has 9 pins on its head

Table 1. A table defining the task of each pin on a DB-9 serial cable

Pin #	Label	Signal Name	Signal Type
1	CD	Carrier detect	Control
2	RD	Received data	Data
3	TD	Transmitted data	Data
4	DTR	Data terminal ready	Control
5	GND	Signal ground	Ground
6	DSR	Data set ready	Control
7	RTS	Request to send	Control
8	CTS	Clear to send	Control
9	RI	Ring indicator	Control

The Boe-Bot uses RS-232 serial communication, which uses the DB-9 serial cable to transmit data back and forth. The Figure above shows how the DB-9 cable is designed, and the Table above shows the specific task each pin is assigned in serial communications.

Yan-Fang Li, in his **Matlab-Based Graphical User Interface Development for Basic Stamp 2 Microcontroller Projects**, says "serial communication is a low-level protocol used for data communication between two or more devices. As the name implies, serial communication uses a data port to send/receive data in a serial manner, i.e., one bit at a time." Therefore, it is required that the communicating devices must operate at the same communication rate, or baudrate. For the serial communication portion we set the baudrate to 9600 bps.

To establish communication between the Boe-Bot and MATLAB, some basic programming commands must be used in both PBASIC and MATLAB.

PBASIC Commands:

SERIN Rpin,baudmode,{Timeout,Tlabel},[InputData]

This function allows the Basic Stamp 2 to receive data through the serial connection. Rpin is a variable that specifies the I/O pin where the data will be received. For wired communication, this pin will always be set as 16. But, for wireless communication it can be any value from 0-15, dependant on where the Rx pin is plugged in. Baudmode defines at what baudrate the communication will operate at. The Basic Stamp 2 can handle baudrates up to 50,000 bps. For the serial communication we used a baudrate of 9600, but once we started working with the Surelink wireless antenna, we

began to use a baudrate of 38400. Timeout gives a time, in milliseconds, for which the program will wait for data to arrive through the serial connection. Tlabel tells the program where to go in the event that a timeout is reached and SERIN is aborted. InputData is a series of variables and formatters that organize the incoming data.

SEROUT Tpin,Baudmode,[OutputData]

This function allows the Basic Stamp 2 to transmit data to another device. Tpin is like Rpin from SERIN, in that it defines the pin in which data will be sent out of the Stamp. For serial communication, this is again set to 16, and for wireless it can be defined as any value from 0-15. Baudmode sets the baudrate at which the data is sent. To insure steady communication this must be equal to the value from SERIN. OutputData is a series of variables and formatters of all data that is being sent to another device.

MATLAB Commands:

ser_obj = serial('COM Port','baudrate',baudrate)

This command defines a serial object ser_obj, as a specific serial port operating at a specified baudrate. The terminator for the file communication, as well as the timeout value can be defined by adding .terminator or .timeout to the end of ser_obj.

fopen(path)/fclose(path)

These commands are used mostly in file communications to open or close a specific file. However, they can also be used to open or close the COM port defined by ser_obj.

freeserial(serial)

This command frees the specified serial port so that it can be used by other programs.

[a,b,c] = fscanf(ser_obj,format)

This command reads in data from the COM port specified by ser_obj. The variable a reads in the actual value, the variable b reads the size of the data, and the variable c stores the default error message. The format of the data in a can be changed by changing the format command. The format can range from a character format to a double format, to ASCII values.

fprintf(ser_obj,format,data)

This command sends data to the COM port specified by ser_obj. Once again, the format of the data being sent can be altered by changing the format modifier.

After we built serial communications protocols, we began to work on wireless communications using the Surelink wireless antenna. The first thing we had to do was to configure the Surelink and to integrate it into the Boe-Bot.

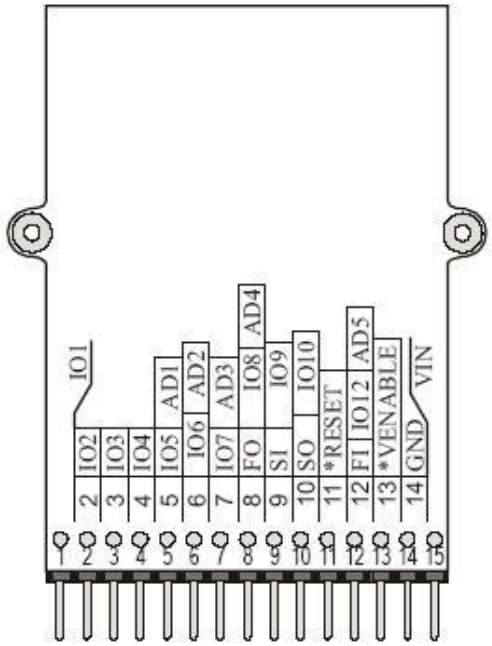


Figure 2. A Surelink RF Antenna, with pin numbers shown

Table 2. Table showing the specific tasks of each pin in the Surelink RF Module

Pin #	Function	Aux Function	QuickLink Jumper	Jumper Position	Description
Pin 1			'M'	High = command mode	Mode Select
				Low = Demo & Cable Link Mode	
				Removed for all other modes	
Pin 2			'S'	High = 4800 bps	RF Data Rate
				Low = 76800 bps	
				Removed = 19200 bps	
Pins 3,4			'BD'	See **Figure 11**	Serial Baud Rate
Pins 5,6		AD1, AD2	'CH'	See **Figure 12**	RF Channel
Pin 7		AD3	'MS'	Low = Point A Unit	Point A/Point B
				High = Point B Unit	
Pin 8	FO	AD4			Flow Control Output
Pin 9	RX				Data Input
Pin 10	TX				Data Output
Pin 11	RESET				Active Low reset
Pin 12	FI	AD5			Flow Control Input
Pin 13	VEN				Must be grounded
Pin 14	GND				Ground
Pin 15	VIN				+5V Supply voltage

Table 3. Table defining the different jumper positions for PIN3 and PIN4 depending upon the specified baudrate

Jumper Position		Baud	Miscellaneous
PIN3	PIN4	Rate	Notes
Removed	HIGH	115200	Default for Cable Link and Command Mode
LOW	LOW	57600	
LOW	Removed	38400	
HIGH	Removed	19200	
Removed	LOW	9600	
HIGH	LOW	2400	
Removed	Removed	1200	Default for all other modes
LOW	HIGH	300	
HIGH	HIGH	N/A	Default for Demo Mode

Table 4. Table defining the jumper positions for PIN5 and PIN6 depending upon the specified RF Channel

Jumper Position		RF
PIN5	PIN6	Channel
LOW	LOW	0
LOW	Removed	1
Removed	LOW	2
Removed	Removed	3
LOW	HIGH	4
Removed	HIGH	5
HIGH	LOW	6
HIGH	Removed	7
HIGH	HIGH	15

The Surelink RF antenna is a highly versatile wireless communications antenna. It has 15 pins on it, and each of these pins has a specific task in wireless communications. These tasks are defined in Table 3. For the task of communicating between the Boe-Bots and the computers, we needed six Surelink antennas. After each of the antennae were formatted into cable link mode using the Surelink Control program provided with the

antennae, the Surelink Modules were ready to be set up. One each was plugged into the three Boe-Bot's breadboards, with circuits designed for power and communications. The other three were inserted into Quicklink Demo Boards and attached by USB-to-Serial adapters to each computer. The set up of the jumpers on the Quicklink Demo Board is defined in Tables 4 and 5. The jumper settings are dependent upon the baudrate and RF channel that the Surelink Antenna is operating on.

Figure 3. A Quicklink Demo Board that is used in our experiment

Figure 4. How the Surelink Antenna and the Quicklink Demo Board are used together for wireless communications

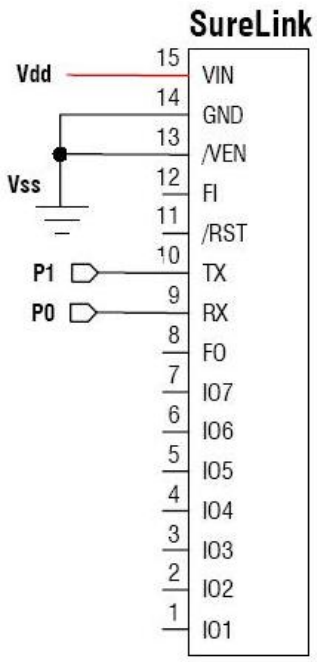


Figure 5. A circuit diagram for the Surelink RF Antennae that were plugged into the Boe-Bot

We found that the software portion of the protocol for wireless communications was remarkably similar to that of the serial communications. All of the same commands were used in PBASIC and MATLAB, with changes being made only to the Tpin and Rpin, and baudrate values on the PBASIC side, and the baudrate values only on the MATLAB side. The Tpin and Rpin were no long 16, but 1 and 0 respectively for our robots. Also, we increased the baudrate to 38400 bps because this would insure much faster decision making abilities for the robot, since data is being sent to and from the computer much faster.

Bibliography

[1] Kai, W. O., Seet, G., Siang, K. S., "Sharing and Trading in a Human-Robot System", Cutting Edge Robotics, Advanced Robotic Systems, 2005, pp467-496.

[2] Pike, J., "Special Weapons Primer", Federation of American Scientists, http://www.fas.org/nuke/intro/missile/basics.htm

[3] Chou, J. H., "Automatic Guided Vehicle", International IEEE/IAS Conference, Industrial Automation and Control: Emerging Technologies, May 1995, pp241-245.

[4] "Testing American Ingenuity in Autonomous Vehicle Design", Defense Advanced Research Projects Agency (DARPA), http://www.darpa.mil/grandchallenge/

[5] Van de Vegte, J., *Feedback Control Systems*, 3rd ed., Prentice Hall, UK, 1993.

[6] Nise, N. S., *Control Systems Engineering*. 3rd ed., Wiley, New York, 2000.

[7] Dayan, P., Abbott, L. F., *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, 1st ed., The MIT Press, Cambridge, MA, 2001.

[8] Kecman, V., *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*, The MIT Press, Cambridge, MA, 2001.

[9] Peggy, T., "Science and Technology - Artificial Intelligence", Thomson Gale Corporation, http://www.scienceclarified.com/scitech/Artificial-Intelligence/

[10] Gurney, K., *An Introduction to Neural Networks*, Routledge, London, 1997.

[11] Sutton, R. S., Barto, A. G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[12] Genevieve Orr, "Neural Networks", Willamette University, http://www.willamette.edu/~gorr/classes/cs449/intro.html

[13] McInroy, J. E., Wilamowski, B. M., "Bipolar Pattern Association Using A Recurrent Winner Take All Network", International Conference on Neural Networks (ICNN), 1997, vol. 2, pp1231-1234.

[14] "Measurement and Automation Software", National Instruments, http://www.ni.com/software/

[15] "Machine Vision", National Instruments, http://www.ni.com/vision/

[16] "Matlab and Simulink for Technical Computing", The MathWorks, http://www.mathworks.com/

[17] "Creative Webcam NX Pro", Creative WorldWide, http://www.creative.com/-products/product.asp?category=218&subcategory=219

[18] "Q-See 2.4G Mini Wireless Camera", Digital Peripheral Solutions Inc., http://www.qps-inc.com

[19] "Composite Video", http://en.wikipedia.org/wiki/Composite_video

[20] "DVD Maker USB 2.0", http://www.kworld.com.tw/en/product/editing/-002/dvd_maker_usb2.0.htm

[21] "NI-IMAQ for USB Cameras User Guide", National Instruments, January 2005.

[22] "Vision Assistant: Interactive Software for Vision Application", National Instruments, http://www.alliancevision.com/us/products/software_ni/-vision_assistant.htm

[23] Lee-Johnson, C. P., "The Development of a Control System for an Autonomous mobile Robot" University of Waikato, 2004.

[24] "SureLink 900 MHz RF Module (#30065), QuickLink Demo Board (#30066)", Parallax Inc., Rev 1.3, October 2004.

[25] Lindsay, A., Robotic with the Boe-Bot, Parallax Inc., Ver. 2.2, 2004.

The vita has been removed from the reformatted version of this document.